［新装版］

X1

シリーズ全機種に対応!!

マシン語活用百科

清水保弘

秋葉出版

X1

マシン語活用百科

X1シリーズ全機種対応！

清 水 保 弘

秋葉出版

清水保弘

マシン語活用百科

秋葉出版

まえがき

パーソナルコンピュータ———それは１つの美しい小宇宙にも比すべき世界です。机の上に載るほどの機械の中に、20世紀後半の人類が到達した叡知が凝縮されているのを眼のあたりにして、感動を覚えるのは筆者だけではないでしょう。

本書は、シャープの開発した「Ｘ１シリーズ」という素晴らしい８ビットパーソナルコンピュータ(パソコン)の〝内部宇宙〞を探険するための案内書です。

パソコンに指令を与えるプログラミング言語として、私たちは普通、BASIC言語を用いています。Ｘ１シリーズにおいても、SHARP HuBASICという高機能のBASICが基本システム言語として用意されていて、手軽にＸ１の素晴らしさを体験することができます。

しかし、Ｘ１シリーズの本当の「底力」——これこそＸ１の真の姿——を見るためには、「マシン語」(機械語) という神秘のベールの彼方に進まねばなりません。本書は、あの冒険活劇映画 「インディ・ジョーンズ」のように、Ｘ１シリーズの内部深くに秘められた数々の「宝」を皆さんが探しあてるヒントを提供しています。

Ｘ１シリーズをマシン語で操作できるようになると、BASICでは思いもよらなかったことが可能になります。たとえば、AUTOやLIST命令の実行時に背景のグラフィック画面を消さないようにするといった小さな工夫から、画面を１ドット単位の細かさでプリンタに「ハードコピー」するといったことまで実現することができます。また、フロッピーディスクのユーザーの方々の中には、ディスクBASICの文法の面倒臭さを感じておられる方もいらっしゃるでしょう。マシン語サブルーチンによるディスクの制御法を知ると、要するに「ディスクを読む」、「ディスクに書き込む」という基本処理さえ理解すれば大抵のことができるのだということが改めて納得できます。

本書は、ユーザーの立場に徹して、Ｘ１シリーズをマシン語で操作するためのこうした「小さなアイデア」を満載してあります。

Ｘ１シリーズには、Ｚ80ＡというLSIがメインCPUとして使われていますから、Ｘ１をマシン語で操作するには、Ｚ80Ａのマシン語を理解する必要があります。さいわいＺ80Ａ(ないしはＺ80)は８ビットのパソコンで最も多く使われているCPUですから 解説書も数多く出版されています。そこで本書では、Ｚ80Ａのマシン語を基礎から説明する部分は思い切って割愛し、これを応用してＸ１シリーズをマシン語で動かす部分に的をしぼりました。Ｚ80マシン語にまだ不慣れな方は、巻末の「参考文献」であげた本などを参照してください。本書のプログラムは、Ｚ80マシン語を「Ｘ１シリーズで動かしながら」学習するための格好の応用例題となることでしょう。本書が、Ｘ１シリーズを愛するすべてのユーザーの皆様の座右の書となることを願ってやみません。

1984年11月　　　清　水　保　弘

目　次

第0章　マシン語プログラミングにあたって

0-1　X1シリーズにおけるI／O空間

X1シリーズの本体には、CPUとしてシャープ製の

LH 0080 A

が搭載されています。このLSIはZ80Aと全く同じもので、シャープがザイログのセカンド・ソースとしてZ80Aを製造する時の型番です。X1では、Z80Aを4MHzのクロック周波数で働かせています。

Z80Aは、データ・バス8ビット、アドレス・バス16ビットを持ち、64Kバイトのメモリーをアクセスすることができます。これを称して「**メモリー空間**（memory field）の容量は64Kバイトである」といいます。

Z80Aはメモリー空間とは別に、各種の入出力装置(I／O)との信号のやりとりをするためアドレスをつけてアクセスできる領域――**I／O空間**(I／O field)――を持っています。通常、「Z80AのI／O空間の容量は256バイトである」といわれているし、多くの本でもこのように書かれていますが、実は、「Z80AのI／O空間の容量は64Kバイトである」というのが本当のところです。

たとえば、出力命令として

OUT　(n)，A

という形のものがあります。通常これは、「n番のI／OポートにAレジスタの内容を出力する」と読み、実際NECのPCシリーズや、シャープのMZシリーズでもそのような使い方をしているのですが、実はこのとき、Aレジスタの内容がアドレス・バスの上位8ビットに出力されるのです。上記のパソコンのようにZ80Aの「通常の」使用法では、入出力命令におけるアドレス・バス上位8ビットを利用していないために「n番のポートに…」の記述で十分な訳ですね。そして、実際にZ80Aのマシン語体系もそのような使用法を前提として設計されています（ニーモニック表記にもそれが現われています)。

ところが、X1シリーズでは入出力命令におけるアドレス・バス16ビット情報をフルに活用する設計をしています。このために、画面表示関係のメモリーであるビデオRAM全部をI／O空間に配置することが可能となり、メイン・メモリーをすべてRAMとするシャープの「クリーン設計」の1つの到達点となったのでした。

従って、X 1シリーズにおいて、出力命令 OUT (00 H), A はたとえばAレジスタの内容が 40 Hのときには、「I／Oポートの 4000 H 番地にデータ 40 Hを出力する」という意味になるので注意しましょう。本書では、 OUT (n), A の機能を象徴的に

I／O (An) ← A

と書くことにします。 LD (HL), A の機能を (HL) ← A と略記することがありますが、その連想として御理解下さい。同様に、入力命令 IN A, (n)の機能はA ← I／O(An)と書かれます。

入出力命令にはこの他に OUT (C), A と IN A, (C) の形のものがあります。この場合は、Bレジスタの内容がアドレス・バスの上位8ビットに出力されるので、各々 I／O (BC) ← A, A ← I／O(BC) という機能になります。

0-2 マシン語プログラミングでの注意点

X 1シリーズにおけるこうしたI／O空間の利用は、反面、マシン語プログラムにおけるいくつかの制約を作り出します。

たとえば、Bレジスタをカウンタとして用いる条件ジャンプ命令 DJNZ e を使う時には、 OUT (C), A の形の命令に注意を払わなくてはなりません。

```
        LD B, n         ; カウンタ設定
LOOP:   PUSH BC         ; カウンタ保護
          ⋮
        LD  BC, ml
        OUT (C), A      ; I／O (ml) ← A
          ⋮
        POP BC          ; カウンタ復帰
        DJNZ LOOP
```

のように、I／Oポートのアドレス指定でのBレジスタと、カウンタとしてのBレジスタを区別しなくてはなりません。

またZ 80 Aの命令セットの中には、ブロック入出力命令

INI, INIR, IND, INDR
OUTI, OTIR, OUTD, OTDR

が用意されていますが、これらはいずれもBレジスタをカウンタとして用いるために、X 1シリーズでのマシン語プログラミングに際して使いにくい命令であると言えましょう。

以上、I／O空間に関連した注意点を挙げましたが、もう1つ、割り込みに関しても注意しなくてはなりません。

CPU 停止命令として HALT があります。PC 系のパソコンでマシン語プログラミングされた方なら経験がありましょうが、マシン語プログラムの末尾に置いて、プログラムの停止をするのに用いることがあります。ところが、X 1において HALT では、プログラムの止まらない事が往々にしてあります。実際、私もマシン語学習の初期に「原因不明の暴走」を何回も経験しましたが、多くの場合、犯人は HALT のようでした。

HALT を実行すると、CPU は命令の進行を止め、リセット信号か割り込み信号が入力されるまで待機状態になる訳ですね。この「割り込み」がクセ物です。詳細は第 7 章を御覧いただきたいのですが、X 1シリーズにおいては、キー入力を割り込みにより処理している関係で、CPU には頻繁に割り込みがかかっているはずです。そこで、 HALT による待機が解除されてしまうのだ！　と私は推測します。(もっと他の原因があるかもしれません。もし御存知の方がありましたら、御教示いただければ幸です。)

こういう訳で、X 1シリーズにおいてはマシン語プログラムの末尾を HALT で終えるのは「タブー」としておくのが安全です。

また、同様の理由から、割り込み処理に関連する命令(DI、EI、IM 関係の命令、 I レジスタ関係の命令) をユーザーが勝手にいじると、キー入力が効かなくなることがありますから、注意が必要です。

0－3　X 1シリーズで利用できるアセンブラ

マシン語学習の初期においては、「ハンドアセンブル」というのも大切な練習の 1 つのようです。実際、私も「ハンドアセンブル」により随分「マシンコード」に対する理解が深まったように思います。本書においても、「アセンブラ風のハンドアセンブル」として、BASIC の DATA 文でマシンコードを格納し、REM 文で、ニーモニックを添える形式のプログラムが沢山登場しますが、これは私の「ハンドアセンブル」の体験から生み出された形式です。(BASIC のスクリーン・エディット機能を用いて、画面を見ながらニーモニックをキー・インし、これらが一通り終わったら、その前に DATA 文として、マシンコードを書き込んでいくものです。これだと紙を使いません。)

短いプログラムなら、これで十分いけると思いますが、長いものではさすがに相対アドレスの計算や、絶対ジャンプ先のアドレス決定など、ウンザリしてきます。こうなると「アセンブラ」の登場です。

ところが最初の頃、X 1にはアセンブラがありませんでした。昔からの MZ や PC に「マシン語ツール」が完備している状況と比べると「悲惨」といってよい状況でした。こうした中で恐らく初めて公表されたアセンブラが「ミニアセンブラ」とよばれるものです。

ミニアセンブラ　(dB SOFT)
I／O 誌　1983 年 3 月号　353 ページ～355 ページ

私もこのアセンブラには随分助けてもらいました。前著『X 1マシン語入門』のサンプル・ゲームは、このアセンブラで作成しました。しかし、段々に不満もたまってきました。16 進数を $ で表記すること（個人的趣味の問題ですが、私は……Hの方が好きです。）、

また、原因不明のアセンブル・エラーを時々生ずること(16ビットのレジスタを扱う時によく出ました。たとえば EX DE, HL や EXX が変換されない例は何回も経験しました。)、アドレスのオフセット（オブジェクト作成アドレスと、アセンブル・リストのアドレスを別々にできる機能）ができないことなどは不満のうちの代表的なものです。自分でアセンブラを改造するか、製作すれば BEST なのですが、初心者の私にはその実力がありませんでした。

こうして待つうちに、事態は飛躍的に改善されました。１つは、(株) 計測技研から「ソフトウェアコンバーター」シリーズが発売されたこと、もう１つは、Ｘ１用 CP／M の発売です。

『マイコン』誌などでも高橋雄一氏によりいく度か紹介されましたが、(株)計測技研の「システムソフトウェアコンバーター」（商品コード B 6-2213）を用いると、Ｘ１が MZ-2000 に変身してしまうのです。こうして、MZ-2000 用の BASIC システムテープをロードすると、それをＸ１用に自動的に改造するというすごいソフトです。そして、さらに驚くべきことに、一部の MZ-2000, MZ-80 B 用マシン語プログラムも動くというのです。これについて詳細を発表して下さったのが松木透氏で、雑誌記事

「Ｘ１マシン語入門　　システムソフトの活用」
Oh！MZ　1983 年 11 月号　80 ページ～88 ページ

の中で、「システムソフトウェアコンバーター」の活用法を紹介しておられます。この記事により、MZ-80 B，MZ-2000 用のアセンブラである

EDITOR ASSEMBLER　　EA-MZ
（アスキー製）

がＸ１でも動かせることを知りました。現在、私はこのアセンブラを愛用しています。本書のソース・プログラムの多くは、EA-MZ を使用して作成されました。

次に CP／M の話をします。[注]Ｘ１用 CP／M の発売により、Ｘ１シリーズは新しい段階に入ったとも言えるでしょう。CP／M のトランジェント・コマンドとして付属してくる ASM（8080 ニーモニックのアセンブラ）、DDT（デバッガ）を用いると、DDT のトレース機能を用いて、１ステップ毎のレジスタの変化を追うことができます。また、少々高価ですが、MAC や MACRO-80 など Z 80 ニーモニックのアセンブラや、SID, ZSID などのデバッガを利用すると、本当に「システムの開発」という感じがしてきます。これらに関しては私の利用経験も浅いので、本書ではこれ以上触れません。

CP／M が余りにも有名なので、その裏に隠れているようですが，Ｘ１ではもう１つの「DOS」を動かすことができます。綜合情報システム研究所から発売されている AGOS(アゴス) という DOS です。この上で、エディタ・アセンブラ EDAM や、デバッガ ZDBG が動きます。これについては、『AGOS 活用ブック』(綜合情報システム研究所・アジェンダグループ編。オーム社刊) を参照して下さい。

これらの状況を見ると、現在では、Ｘ１シリーズをとりまく「開発ソフトウェア」は次第に充実してきていると言えそうです。

[注] CP／M は、デジタルリサーチ社の登録商標です。

0－4　アセンブラ・ソースリストの見方

前節で説明したように、本書のプログラムリスト(ソースリスト)の多くは、EA-MZを用いて作成されていますので、簡単にリストの見方を説明しておきます。

ソースリストの例（1）

```
0000                  ;**********************************************************
0001                  ;*                                                        *
0002                  ;*      main routine for PCG set (example)                *
0003                  ;*                                                        *
0004                  ;**********************************************************
0005                  ;
0006                  ;
0007 E000             PCGSUB:EQU  0E000H          ;PCG data set subroutine
0008 1000             MONHOT:EQU  1000H           ;Hu BASIC monitor hot start
0009                  ;
0010                  ;
0011                         ORG  0D000H
0012                  ;
0013                  ;
0014                  ;----- initial setting -----
0015                  ;
0016 D000 1630               LD   D,30H           ;D  = start ASCII code for PCG set
0017 D002 1E15               LD   E,15H           ;E  = PCG select code (Blue)
0018 D004 211CD0             LD   HL,DATA         ;HL = top address of PCG data
0019 D007 0604               LD   B,4             ;B  = counter of PCG set (4 char)
0020                  ;
0021                  ;
0022                  ;----- main loop for PCG set -----
0023                  ;
0024 D009 D5          LOOP:  PUSH DE              ;Keep D,E
0025                  ;
0026 D00A CD00E0             CALL PCGSUB          ;blue pattern set
0027 D00D 1C                 INC  E               ;PCG select code = 16H
0028 D00E CD00E0             CALL PCGSUB          ;red pattern set
0029 D011 1C                 INC  E               ;PCG select code = 17H
0030 D012 CD00E0             CALL PCGSUB          ;green pattern set
0031                  ;
0032 D015 D1                 POP  DE              ;D,E back (E = 15H)
0033 D016 14                 INC  D               ;to next character
0034 D017 10F0               DJNZ LOOP            ;loop by counter B reg (4 char)
0035                  ;
0036                  ;
0037                  ;----- end of main routine -----
0038                  ;
0039 D019 C30010      EXIT:  JP   MONHOT          ;goto monitor
```

ニーモニック　　コメント

マシンコード(モニターより入力するときは、この部分を入力します。)

アドレス

行番号(ソースリストの行番号で、アセンブラ使用時以外は無視して下さい。)

アセンブラでは、マシンコードに対応するニーモニック以外に、**アセンブラ指示命令(疑似命令)**とよばれる、マシンコードには変換されず、アセンブラへの指示のみ与える命令も使われます。本書のリスト中によく出てくる疑似命令は次のものです。

疑似命令	名　称	機　能
EQU	equate	ラベルに数値を定義する。
ORG	origin	プログラムの開始番地を指定する。
DB	define byte	データ列をメモリーに格納する。
DW	define word	2バイトデータを上下位逆転してメモリーに格納する。
DEFM	define message	メッセージ（文字列）をメモリーに格納する。
DS	define storage	ある範囲のメモリーを確保する。
END	end	アセンブラにプログラム終了を指示する。

上記の例では、第7行は「E 000 H という 16 進数に PCGSUB というラベルをはる」の意味です。以後、アセンブラは PCGSUB をすべて E 000 H でおきかえてくれます。第11行は「プログラムを D 000 H 番地から格納する」の意味です。第24行には、アドレスに直接 LOOP というラベルをつけました。ですから、34行の DJNZ LOOP は、ここにジャンプすることを意味します。

ソースリストの例（II）

```
0040                    ;
0041                    ;
0042                    ;----- PCG pattern data -----
0043                    ;
0044                    ;***** data for code 30H *****
0045                    ;
0046 D01C 01020408 DATA:  DB    01H,02H,04H,08H,10H,20H,40H,80H
     D020 10204080
0047 D024 01020408        DB    01H,02H,04H,08H,10H,20H,40H,80H
     D028 10204080
0048 D02C 01020408        DB    01H,02H,04H,08H,10H,20H,40H,80H
     D030 10204080
0049                    ;
0050                    ;***** data for code 31H *****
0051                    ;
0052 D034 01020408        DB    01H,02H,04H,08H,10H,20H,40H,80H
     D038 10204080
0053 D03C 00000000        DB    00H,00H,00H,00H,00H,00H,00H,00H
     D040 00000000
0054 D044 00000000        DB    00H,00H,00H,00H,00H,00H,00H,00H
     D048 00000000
0055                    ;
0056                    ;***** data for code 32H *****
0057                    ;
0058 D04C 00000000        DB    00H,00H,00H,00H,00H,00H,00H,00H
     D050 00000000
0059 D054 01020408        DB    01H,02H,04H,08H,10H,20H,40H,80H
     D058 10204080
0060 D05C 00000000        DB    00H,00H,00H,00H,00H,00H,00H,00H
     D060 00000000
0061                    ;
0062                    ;***** data for code 33H *****
0063                    ;
0064 D064 00000000        DB    00H,00H,00H,00H,00H,00H,00H,00H
     D068 00000000
0065 D06C 00000000        DB    00H,00H,00H,00H,00H,00H,00H,00H
     D070 00000000
0066 D074 01020408        DB    01H,02H,04H,08H,10H,20H,40H,80H
     D078 10204080
0067                    ;

0068                    ;
0069 D07C                 END
```

上がソース・リストの終わりの部分です。疑似命令 DB により、指定されたデータがメモリーに格納されている様子がわかりますね。第69行は疑似命令 END です。アドレス値は出力されますが、マシンコード欄は空ですね。従って、実際のプログラムはＤＯ７ＢＨ番地で終わることになります。

以上、リストの見方の概略を説明しておきました。あとは各章で実例を見ながら、適宜考えて下さい。

第1章　テキスト画面

1－1　X1シリーズの画面表示機能

X1シリーズの機能の素晴らしさの筆頭に来るのが、グラフィック機能でありましょう。

X1シリーズは、テキスト画面とグラフィック画面という2種類の画面表示をすることができます（最初のX1――CZ-800C――のみ、グラフィック画面表示はオプションになっています)。「グラフィック」という本来の意味と直結するグラフィック画面表示機能はもちろん高水準なのですが、驚くべきはテキスト画面表示機能の充実ぶりです。とくに256文字をユーザーがドット単位で定義できる機能（PCG）は、X1の発表当時ずば抜けていて、テキスト画面だけでも、精緻な画面を持ったゲームが作れるほどでした。また、倍文字表示機能は、テレビ画面とのスーパー・インポーズ（重ね合わせ表示）時に字幕などを入れることを意図して作られているようで、X1シリーズの多方面への利用を可能にしています。

本章では、こうした豊富な機能を持つテキスト画面を扱います。

1－2　テキストVRAM

テキスト画面への文字表示は、テキストVRAMに文字のコードを送ることにより行なわれます。

テキストVRAMは、テキスト画面表示専用のメモリー（VRAM＝Video RAM）で、X1シリーズにおいては、I／O空間内の3000H番地から37FFH番地に配置されています。テキストVRAMの各I／Oアドレスは、テキスト画面の表示位置と対応しています。40桁表示（横40字×縦25字)、80桁表示（横80字×縦25字）の各々で対応の仕方は異なりますが、図示すると以下のようになります。

40 桁表示画面ページ 0 の VRAM アドレス

3000H	3001H	3002H	3003H	3004H	3005H	3006H			3025H	3026H	3027H
3028H	3029H	302AH	302BH	302CH	302DH	302EH			304DH	304EH	304FH
3050H	3051H	3052H	3053H	3054H	3055H	3056H			3075H	3076H	3077H
3078H	3079H	307AH	307BH	307CH	307DH	307EH			309DH	309EH	309FH
30A0H	30A1H	30A2H	30A3H	30A4H	30A5H	30A6H			30C5H	30C6H	30C7H
30C8H	30C9H	30CAH	30CBH	30CCH	30CDH	30CEH			30EDH	30EEH	30EFH
30F0H	30F1H	30F2H	30F3H	30F4H	30F5H	30F6H			3115H	3116H	3117H
3370H	3371H	3372H	3373H	3374H	3375H	3376H			3395H	3396H	3397H
3398H	3399H	339AH	339BH	339CH	339DH	339EH			33BDH	33BEH	33BFH
33C0H	33C1H	33C2H	33C3H	33C4H	33C5H	33C6H			33E5H	33E6H	33E7H

40 桁表示画面ページ 1 の VRAM アドレス

3400H	3401H	3402H	3403H	3404H	3405H	3406H			3425H	3426H	3427H
3428H	3429H	342AH	342BH	342CH	342DH	342EH			344DH	344EH	344FH
3450H	3451H	3452H	3453H	3454H	3455H	3456H			3475H	3476H	3477H
3478H	3479H	347AH	347BH	347CH	347DH	347EH			349DH	349EH	349FH
34A0H	34A1H	34A2H	34A3H	34A4H	34A5H	34A6H			34C5H	34C6H	34C7H
34C8H	34C9H	34CAH	34CBH	34CCH	34CDH	34CEH			34EDH	34EEH	34EFH
34F0H	34F1H	34F2H	34F3H	34F4H	34F5H	34F6H			3515H	3516H	3517H
3770H	3771H	3772H	3773H	3774H	3775H	3776H			3795H	3796H	3797H
3798H	3799H	379AH	379BH	379CH	379DH	379EH			37BDH	37BEH	37BFH
37C0H	37C1H	37C2H	37C3H	37C4H	37C5H	37C6H			37E5H	37E6H	37E7H

80 桁表示画面の VRAM アドレス

3000H	3001H	3002H	3003H	3004H	3005H	3006H			304DH	304EH	304FH
3050H	3051H	3052H	3053H	3054H	3055H	3056H			309DH	309EH	309FH
30A0H	30A1H	30A2H	30A3H	30A4H	30A5H	30A6H			30EDH	30EEH	30EFH
30F0H	30F1H	30F2H	30F3H	30F4H	30F5H	30F6H			313DH	313EH	313FH
3140H	3141H	3142H	3143H	3144H	3145H	3146H			318DH	318EH	318FH
3190H	3191H	3192H	3193H	3194H	3195H	3196H			31DDH	31DEH	31DFH
31E0H	31E1H	31E2H	31E3H	31E4H	31E5H	31E6H			322DH	322EH	322FH
36E0H	36E1H	36E2H	36E3H	36E4H	36E5H	36E6H			372DH	372EH	372FH
3730H	3731H	3732H	3733H	3734H	3735H	3736H			377DH	377EH	377FH
3780H	3781H	3782H	3783H	3784H	3785H	3786H			37CDH	37CEH	37CFH

（注）表示桁数の指定は、HuBASIC では WIDTH 40 あるいは WIDTH 80 で行なわれます。

（注）40 桁表示の時には、テキスト画面を 2 枚持つことができます。各々、ページ 0，ページ 1 とよぶことにします。HuBASIC で、ページ切り替えをするには、SCREEN 文が用いられます（マニュアル参照）。

HuBASICでは、テキスト画面の表示位置は通常、x、y座標によって指定されます(LOCATE文)。座標とVRAMアドレスとの換算式は以下の通りです。

テキスト画面	換　算　公　式
40桁画面ページ0	(VRAMアドレス)=&H3000+(x座標)+(y座標)*40
40桁画面ページ1	(VRAMアドレス)=&H3400+(x座標)+(y座標)*40
80　桁　画　面	(VRAMアドレス)=&H3000+(x座標)+(y座標)*80

1-3　キャラクタ・コード(ASCIIコード)

テキストVRAMには、文字自体ではなく、その文字に対応するキャラクタ・コード(ASCIIコード)が格納されます。X1シリーズにおいては、右表のようなコードが用いられます。

たとえば、アルファベット大文字のAは、16進表記で41H(あるいは&H41)というキャラクタ・コードを持っています。00H～1FHまでのコードは、コントロール・コード といって、改行、カーソル移動、画面消去、ベルなどの働きに対応します。

キャラクタ・コード表

上位4ビット→

下位4ビット↓

0123456789ABCDEF

1-4　テキスト属性(アトリビュート)

文字をテキスト画面にどのように表示するか(たとえば、色は何色にするか、白黒反転させるか、点滅させるか等)を指定するためのデータを、**テキスト属性**とよびます。属性のことを英語で、attribute(アトリビュート)というので、テキスト属性を単に**アトリビュート**と略称したりします。

X1シリーズの強力な点として、各文字ごとに独立にアトリビュート指定できることが挙げられます。アトリビュート指定は、1バイト(8ビット)のデータで行なわれ、その各ビットは次の意味を持っています。(各ビットには意味にちなんだ名称がつけられています。たとえばHSはHorizontal Size――水平サイズ――から来た名称です。)

アトリビュート・ビットの構成

ビット7	6	5	4	3	2	1	0
HS	VS	CG	CF	CR	G	R	B

CR: 0のとき標準 / 1のとき反転

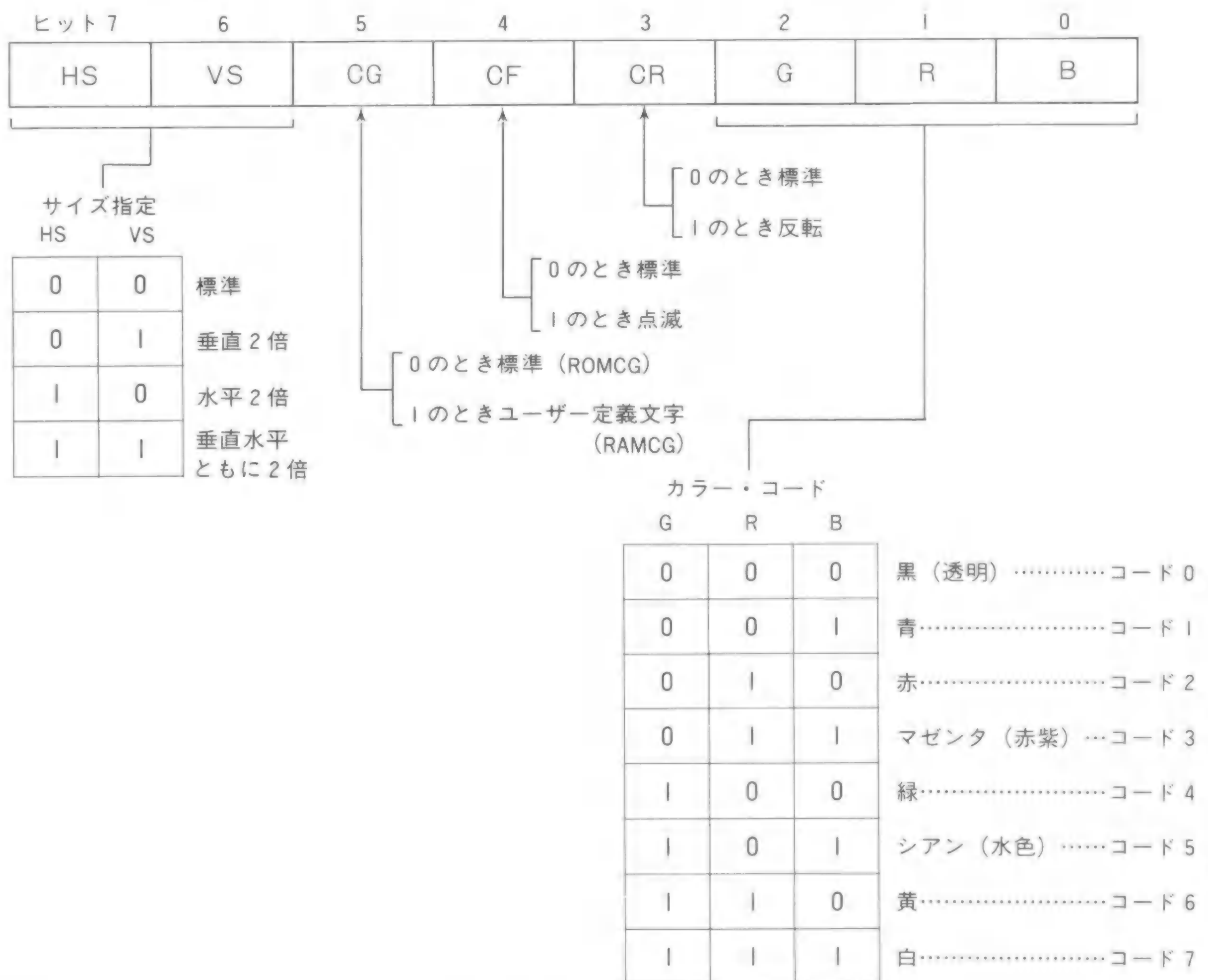
CF: 0のとき標準 / 1のとき点滅

CG: 0のとき標準（ROMCG） / 1のときユーザー定義文字（RAMCG）

サイズ指定

HS	VS	
0	0	標準
0	1	垂直2倍
1	0	水平2倍
1	1	垂直水平ともに2倍

カラー・コード

G	R	B	
0	0	0	黒（透明）…………コード0
0	0	1	青……………………コード1
0	1	0	赤……………………コード2
0	1	1	マゼンタ（赤紫）…コード3
1	0	0	緑……………………コード4
1	0	1	シアン（水色）……コード5
1	1	0	黄……………………コード6
1	1	1	白……………………コード7

［注］ HuBASICでは各アトリビュート・ビットのON／OFFを次のコマンドでサポートしています。

アトリビュート・ビット	HuBASICのコマンド
HS、VS	CSIZE n (n＝0、1、2、3)
CG	CGEN n (n＝0、1)
CF	CFLASH n (n＝0、1)
CR	CREV n (n＝0、1)
G、R、B	COLOR n (n＝0～7)

さて、これらのアトリビュート指定コードは、キャラクタ・コードとは別に、**アトリビュートVRAM（属性VRAM、属性ポート）**とよばれる所に格納されます。これは、I／O空間の2000H番地～27FFH番地に割りつけられていて、テキストVRAMに格納された文字の各々に別々にアトリビュート指定できるように、2つのVRAMアドレスは、次のように1対1に対応しています。

テキストVRAMアドレス	アトリビュートVRAMアドレス
3000H番地	2000H番地
3001H番地	2001H番地
3002H番地	2002H番地
⋮	⋮
37FEH番地	27FEH番地
37FFH番地	27FFH番地

従って、テキスト画面表示の原則は次のようになります。

テキスト VRAM のアドレス　　　　　　　　　←キャラクタ・コード
対応するアトリビュート VRAM のアドレス←アトリビュート・コード

これらの原則さえ理解してしまえば、簡単なリアルタイム・ゲームならマシン語で作成することも容易です。これから数節にわたって画面表示の基本テクニックを解説します。上の原則がどのようにプログラムされているか観察して下さい。

1－5　テキスト画面の1文字表示

まず基本となる1文字表示プログラムを作ってみましょう。40桁モード（ページ0）の画面中央(VRAM アドレス＝31F4H)に黄色で文字Aを表示します。VRAM がI／Oポートに割りつけられていますから、アクセスするために入出力命令を用いている点に御注目下さい。

リスト　　（1文字表示　(1)）

```
0000                    ;**********************************
0001                    ;*                                *
0002                    ;*   1ﾓｼﾞ ﾋｮｳｼﾞ ﾌﾟﾛｸﾞﾗﾑ (1)    *
0003                    ;*                                *
0004                    ;**********************************
0005                    ;
0006                            ORG   0D000H
0007                    ;
0008 D000 01F431                LD    BC,31F4H        ;text VRAM address
0009 D003 3E41                  LD    A,41H           ;ASCII code of "A"
0010 D005 ED79                  OUT   (C),A           ;out to text VRAM
0011 D007 01F421                LD    BC,21F4H        ;attribute VRAM address
0012 D00A 3E06                  LD    A,06H           ;COLOR 6 (yellow)
0013 D00C ED79                  OUT   (C),A           ;out to attribute VRAM
0014 D00E C9                    RET
0015                    ;
0016 D00F                       END
```

このプログラムでは、アトリビュート VRAM アドレスを新たに BC レジスタに設定していますが、前節の VRAM の対応からもわかるように、テキスト VRAM アドレスの上位8ビットであるBレジスタのビット4を0にすれば、BC レジスタはそのまま対応するアトリビュート VRAM のアドレスになります。この方法は、X1のテキスト画面表示でしばしば用いられますから覚えて下さい。

次のプログラムは、今述べたことを用いて「1文字表示プログラム」を書き直したものです。

リスト　　（1文字表示 (2)）

```
0000                    ;******************************
0001                    ;*                            *
0002                    ;*  1ﾓｼﾞ ﾋｮｳｼﾞ ﾌﾟﾛｸﾞﾗﾑ (2)   *
0003                    ;*                            *
0004                    ;******************************
0005                    ;
0006                            ORG     0D000H
0007                    ;
0008 D000 01F431                LD      BC,31F4H        ;text VRAM address
0009 D003 3E41                  LD      A,41H           ;ASCII code of "A"
0010 D005 ED79                  OUT     (C),A           ;out to text VRAM
0011 D007 CBA0                  RES     4,B             ;attribute VRAM address
0012 D009 3E06                  LD      A,06H           ;COLOR 6 (yellow)
0013 D00B ED79                  OUT     (C),A           ;out to attribute VRAM
0014 D00D C9                    RET
0015                    ;
0016 D00E                       END
```

1-6　簡単なアニメーション

1文字表示ができた所で、次に、表示した文字を動かしてみましょう。画面は40桁モード，ページ0とします(もし必要なら、WIDTH 40：SCREEN 0、0にして下さい)。画面のふちを、白色の♥マークにグルグル回ってもらいましょう。

リスト　　（画面1周）

```
0000                    ;*************************
0001                    ;*                       *
0002                    ;*  ﾊｰﾄ ﾏｰｸ ｶﾞﾒﾝ 1 ｼｭｳ   *
0003                    ;*                       *
0004                    ;*      by Y.Shimizu     *
0005                    ;*                       *
0006                    ;*      1984.2.22        *
0007                    ;*                       *
0008                    ;*************************
0009                    ;
0010                    ;----- VRAM address -----
0011                    ;
0012 3000               UPLT:   EQU     3000H           ;ｶﾞﾒﾝ ﾋﾀﾞﾘ ｳｴ ｽﾐ
0013 3027               UPRT:   EQU     3027H           ;ｶﾞﾒﾝ ﾐｷﾞ  ｳｴ ｽﾐ
0014 33E7               DNRT:   EQU     33E7H           ;ｶﾞﾒﾝ ﾐｷﾞ  ｼﾀ ｽﾐ
0015 33C0               DNLT:   EQU     33C0H           ;ｶﾞﾒﾝ ﾋﾀﾞﾘ ｼﾀ ｽﾐ
0016                    ;
0017                            ORG     0D000H
0018                    ;
0019                    ;----- initial setting -----
0020                    ;
0021 D000 16E3          INIT:   LD      D,0E3H          ;ASCII code of heart mark
0022 D002 1E07                  LD      E,07H           ;color 7 (white)
0023                    ;
0024                    ;----- main routine -----
0025                    ;
0026 D004 010030        RIGHT:  LD      BC,UPLT         ;VRAM start setting
0027                    ;
0028 D007 212730        LOOP1:  LD      HL,UPRT         ;corner check
0029 D00A CD7BD0                CALL    CPHLBC
0030 D00D 380E                  JR      C,DOWN          ;if BC > UPLT then go to DOWN
0031 D00F CD85D0                CALL    PRINT           ;print heart mark
0032 D012 D5                    PUSH    DE
0033 D013 CD90D0                CALL    WAIT            ;wait
0034 D016 CD7FD0                CALL    SPACE           ;print space
0035 D019 03                    INC     BC              ;VRAM = VRAM + 1 (right)
0036 D01A D1                    POP     DE
0037 D01B 18EA                  JR      LOOP1
```

```
0038                        ;
0039 D01D 012730            DOWN:   LD    BC,UPRT         ;VRAM start setting
0040                        ;
0041 D020 21E733            LOOP2:  LD    HL,DNRT         ;corner check
0042 D023 CD7BD0                    CALL  CPHLBC
0043 D026 3815                      JR    C,LEFT          ;if BC > DNRT then go to LEFT
0044 D028 CD85D0                    CALL  PRINT           ;print heart mark
0045 D02B D5                        PUSH  DE
0046 D02C CD90D0                    CALL  WAIT            ;wait
0047 D02F CD7FD0                    CALL  SPACE           ;print space
0048 D032 60                        LD    H,B             ;VRAM = VRAM + 40 (down)
0049 D033 69                        LD    L,C
0050 D034 112800                    LD    DE,40
0051 D037 19                        ADD   HL,DE
0052 D038 44                        LD    B,H
0053 D039 4D                        LD    C,L
0054 D03A D1                        POP   DE
0055 D03B 18E3                      JR    LOOP2
0056                        ;
0057 D03D 01E733            LEFT:   LD    BC,DNRT         ;VRAM start setting
0058                        ;
0059 D040 21C033            LOOP3:  LD    HL,DNLT         ;corner check
0060 D043 CD7BD0                    CALL  CPHLBC
0061 D046 3F                        CCF
0062 D047 380E                      JR    C,UP            ;if BC < DNLT
0063 D049 CD85D0                    CALL  PRINT           ;print heart mark
0064 D04C D5                        PUSH  DE
0065 D04D CD90D0                    CALL  WAIT            ;wait
0066 D050 CD7FD0                    CALL  SPACE           ;print space
0067 D053 0B                        DEC   BC              ;VRAM = VRAM - 1 (left)
0068 D054 D1                        POP   DE
0069 D055 18E9                      JR    LOOP3
0070                        ;
0071 D057 01C033            UP:     LD    BC,DNLT         ;VRAM start setting
0072                        ;
0073 D05A 210030            LOOP4:  LD    HL,UPLT         ;corner check
0074 D05D CD7BD0                    CALL  CPHLBC
0075 D060 3F                        CCF
0076 D061 DA04D0                    JP    C,RIGHT         ;if BC < UPLT then go to RIGHT
0077 D064 CD85D0                    CALL  PRINT           ;print heart mark
0078 D067 D5                        PUSH  DE
0079 D068 CD90D0                    CALL  WAIT            ;wait
0080 D06B CD7FD0                    CALL  SPACE           ;print space
0081 D06E 60                        LD    H,B             ;VRAM = VRAM - 40 (up)
0082 D06F 69                        LD    L,C
0083 D070 112800                    LD    DE,40
0084 D073 B7                        OR    A
0085 D074 ED52                      SBC   HL,DE
0086 D076 44                        LD    B,H
0087 D077 4D                        LD    C,L
0088 D078 D1                        POP   DE
0089 D079 18DF                      JR    LOOP4
0090                        ;
0091                        ;----- sub routines -----
0092                        ;
0093 D07B B7                CPHLBC:OR     A               ;reset Cy flag
0094 D07C ED42                      SBC   HL,BC           ;if HL < BC then Cy = 1
0095 D07E C9                        RET
0096                        ;
0097 D07F 1620              SPACE:  LD    D,20H           ;ASCII code of space
0098 D081 CD85D0                    CALL  PRINT           ;print a character
0099 D084 C9                        RET
0100                        ;
0101 D085 7A                PRINT:  LD    A,D             ;A = ASCII code
0102 D086 ED79                      OUT   (C),A           ;out to text VRAM (BC)
0103 D088 CBA0                      RES   4,B             ;BC = attribute VRAM address
0104 D08A 7B                        LD    A,E             ;A = attribute code
0105 D08B ED79                      OUT   (C),A           ;out to attribute VRAM (BC)
0106 D08D CBE0                      SET   4,B             ;BC = text VRAM address
0107 D08F C9                        RET
0108                        ;
0109 D090 210005            WAIT:   LD    HL,500H         ;wait counter initialize
0110 D093 2B                LP:     DEC   HL              ;counter decrement
0111 D094 7C                        LD    A,H
0112 D095 B5                        OR    L
0113 D096 20FB                      JR    NZ,LP           ;wait until count 0
0114 D098 C9                        RET
0115                        ;
0116 D099                           END
```

マシン語プログラムの高速性のため、そのままでは目にもとまらぬスピードになりますから、時間かせぎのサブルーチン（ WAIT というラベルがつけてあります）により、少しだけ動きを遅くしてあります。それでもまだ、かなり高速ですね。

このプログラムは無限ループになっていますので、止めるには背面のリセットボタンを用いて下さい。

1－7　PCGによるアニメーション

前節までは、標準文字を画面表示しましたが、本節では、X１シリーズのテキスト画面テクニックの要とも言うべき**「プログラマブル・キャラクタ・ジェネレータ」**（Programmable Character Generator 略して PCG という）機能を用いて、私たちが自由に作成したキャラクタ・パターン（ユーザー定義文字）を画面表示してみましょう。

キャラクタ・パターンの例として、UFO を描いてみました。下記の BASIC プログラムが、PCG に UFO パターンを登録するものです。

リスト　　UFOキャラクタ

```
100 ' ■■■■■■■■■■■■■■■■■■■■
110 ' ■                  ■
120 ' ■    UFO ｷｬﾗｸﾀ     ■
130 ' ■                  ■
140 ' ■   by Y.Shimizu   ■
150 ' ■    1984.2.25     ■
160 ' ■                  ■
170 ' ■■■■■■■■■■■■■■■■■■■■
180 '
190 INIT :CLS :WIDTH 40
200 LOCATE 12,7 :CFLASH 1 :PRINT "UFO ﾉ PCG ﾃｲｷﾞﾁｭｳ" :CFLASH 0
210 '
220 ' ■■■ ｷｬﾗｸﾀ ｸﾘｱ ■■■
230 '
240 FOR I=0 TO 255
250   DEFCHR$(I)=HEXCHR$("0000000000000000000000000000000000000000000000000000000000000000")
260 NEXT
270 '
280 ' ■■■ ｷｬﾗｸﾀ ﾃｲｷﾞ ■■■
290 '
300 DEFCHR$(&H23)=HEXCHR$("0000000000000F0F000000000000F3F000000000000F2F")
310 DEFCHR$(&H24)=HEXCHR$("0F0F0000000000003F0F000000000002F0F000000000000")
320 DEFCHR$(&H33)=HEXCHR$("0000000000FFFFF0000000000FFFFF000000000FFFFF")
330 DEFCHR$(&H34)=HEXCHR$("FFFF0F00000000FFFF0F0000000009FF00F0000000000")
340 DEFCHR$(&H43)=HEXCHR$("00000000FFFFFFF00000000FFFFFFF00000000FFFFFFF")
350 DEFCHR$(&H44)=HEXCHR$("FFFFFF0F0000000FFFEFE0F0000000FF00F00F0000000")
360 DEFCHR$(&H53)=HEXCHR$("0103070FFFF8F8FF0103070FFFF8F8FF01030509FFFFFFFC")
370 DEFCHR$(&H54)=HEXCHR$("FFFFFCFC0F070301FF0F8CFC0F070301FC0C03F30C040301")
380 DEFCHR$(&H62)=HEXCHR$("000000000030F3FFF00000000030F3FFF00000000030F3FFF")
390 DEFCHR$(&H63)=HEXCHR$("FFFFFFFFFF7F07E1FFFFFFFFFF7F07E1FFFFFFFFFFFFFEFE")
400 DEFCHR$(&H64)=HEXCHR$("F8FF7F7F8F87F8F8F8FF7F7C8C87F8F8FFFFFFFFCFCFFFF9F")
410 DEFCHR$(&H65)=HEXCHR$("FF3F0F0300000000F0380F0300000000C0380C0300000000")
420 DEFCHR$(&H71)=HEXCHR$("0000000003070F1F0000000003070F1F0000000003070F1F")
430 DEFCHR$(&H72)=HEXCHR$("1F1F3FFFFFFFFFFF1F1F3FFFFFFFFFFF1F1F3CFCFFFFFFFF")
440 DEFCHR$(&H73)=HEXCHR$("FFFFFFFFFFFFFFFFFFFFFFFCFC0000C0E0C0FFFCFC0000C00000")
450 DEFCHR$(&H74)=HEXCHR$("3FFFFFFFFFFF7F9F308F8F0000000018C00F0F0000008078")
460 DEFCHR$(&H75)=HEXCHR$("9FFFFFFFFF3F1F1F98E71F00E033131F780707700E0301010")
470 DEFCHR$(&H76)=HEXCHR$("1F0F0400000000001F0F070300000000100C070300000000")
480 DEFCHR$(&H81)=HEXCHR$("00000000C080808000000000C0E0F0F800000000C0E0F0F8")
490 DEFCHR$(&H82)=HEXCHR$("C0C00000C0FCFFFFF8F8FCFFFFFF030018183C3FFFFF0300")
500 DEFCHR$(&H83)=HEXCHR$("FFFFFFFFFFFFFFFFF000000000000000070000000000000007")
510 DEFCHR$(&H84)=HEXCHR$("FFFFFFFFFFFFFFFFF07F0F000000001E07F0F000000001E")
520 DEFCHR$(&H85)=HEXCHR$("FFFFFFFFFF000001EC0C0010FFCF8F81EC0C000000C0838")
530 DEFCHR$(&H86)=HEXCHR$("0000C0C00000000F8F0E0C00000000038F0E0C00000000")
540 DEFCHR$(&H92)=HEXCHR$("00000000C0F0FCFF00000000C0C0F0C400000000C0F0FC03")
550 DEFCHR$(&H93)=HEXCHR$("FFFFFFFFFFFFFFFFC6381C01000E0EC00100000000E0EC0")
560 DEFCHR$(&H94)=HEXCHR$("FFFFFFFFFFFFFFFFFC000000F0FF0F000C000000000F0F000")
570 DEFCHR$(&H95)=HEXCHR$("FFFCF0C00000000033CC30C000000000000000000000000000")
580 DEFCHR$(&HA3)=HEXCHR$("80C0E0F0FFFFFFFF000040608FCF3F0780C02010070F0F06")
590 DEFCHR$(&HA4)=HEXCHR$("FFFFFFFFFF0E0C08020C0D80FD0A0408000000000000000000")
600 DEFCHR$(&HB3)=HEXCHR$("00000000F0FFFFFF00000000F0FFFFCF00000000F0FFCF03")
610 DEFCHR$(&HB4)=HEXCHR$("FFFFFFF000000003CCDFF000000000000003C000000000")
620 DEFCHR$(&HC3)=HEXCHR$("000000000F0FFFF000000000F0FFFF000000000F0FFFF")
630 DEFCHR$(&HC4)=HEXCHR$("FFFFF0000000000FF16F0000000000F0020000000000")
640 DEFCHR$(&HD3)=HEXCHR$("00000000000F0F000000000000F0FC00000000000F0F4")
650 DEFCHR$(&HD4)=HEXCHR$("F0F00000000000FC30000000000000C4300000000000000")
660 '
670 ' ■■■ ｵﾜﾘ ■■■
680 '
690 BEEP
700 LOCATE 12,7 :PRINT "ｵﾜﾘ ﾏｼﾀ｡
710 '
720 END
```

使用するキャラクタ・コードと UFO デザインとの関係は、おおよそ次図のようになっています。

UFO デザイン

では、この UFO パターンを左右に動かしてみましょう。まず、BASIC でプログラムしてみます。次のプログラムを RUN して下さい。

リスト　　UFO move (BASIC)

```
100 REM ***************************
110 REM *                         *
120 REM *     UFO move (BASIC)    *
130 REM *                         *
140 REM *        1984.2.25        *
150 REM *                         *
160 REM *      by  Y.Shimizu      *
170 REM *                         *
180 REM ***************************
190 '
200 WIDTH 40 : INIT : CLS4
210 '
220 REM -- UFO pattern read ------------
230 '
240 DIM UFO$(6)
250 FOR I=1 TO 6
260   FOR J=1 TO 15
270     UFO$(I)=UFO$(I)+CHR$(J*16+I)
280   NEXT J
290 NEXT I
300 '
310 REM -- initial location ------------
320 '
330 X=0 : Y=1 : CGEN 1
340 '
350 LABEL "RIGHT" :REM -----------------
360 '
370 WHILE X<27
380   FOR I=1 TO 6
390     LOCATE X,Y+I-1 : PRINT#0 UFO$(I)
400   NEXT I
410   X=X+1
420 WEND
430 '
440 LABEL "LEFT" :REM ------------------
450 '
460 WHILE X>=0
470   FOR I=1 TO 6
480     LOCATE X,Y+I-1 : PRINT#0 UFO$(I)
490   NEXT I
500   X=X-1
510 WEND
520 '
530 X=0 : GOTO "RIGHT"
540 '
550 REM ------------------------------
```

いかがですか、UFO がノッタリノッタリと歪みつつ左右移動しますね。歪みの原因は、BASIC が遅いので、PRINT 文で上段から下段まで表示するのに時間のズレが生ずるためです。このように多くのキャラクタを表示しようとするとき、BASIC ではよく起こる現象の1つです。

これを解決するために、UFO の左右移動をマシン語で実現します。次のプログラムがその一例です。

リスト　UFO move

```
0000                    ;**************************
0001                    ;*                        *
0002                    ;*       UFO move         *
0003                    ;*                        *
0004                    ;*     by Y.Shimizu       *
0005                    ;*                        *
0006                    ;*      1984.2.25         *
0007                    ;*                        *
0008                    ;**************************
0009                    ;
0010                            ORG   0D000H
0011                    ;
0012                    ;----- clear screen -----
0013                    ;
0014 D000 CD2CD0                CALL  CLS              ;screen clear
0015                    ;
0016                    ;----- main routine -----
0017                    ;
0018 D003 112830        RIGHT:  LD    DE,3028H         ;VRAM address of ( 0,  1)
0019 D006 214230                LD    HL,3042H         ;VRAM address of (26,  1)
0020                    ;
0021 D009 CD45D0        LOOP1:  CALL  CPHLDE           ;2 byte compare
0022 D00C 3809                  JR    C,LEFT           ;if right end then go LEFT
0023 D00E CD4BD0                CALL  UFO              ;print UFO
0024 D011 13                    INC   DE               ;VRAM = VRAM + 1
0025 D012 CD79D0                CALL  WAIT             ;wait
0026 D015 18F2                  JR    LOOP1
0027                    ;
0028 D017 114230        LEFT:   LD    DE,3042H         ;VRAM address of (26,  1)
0029 D01A 212830                LD    HL,3028H         ;VRAM address of ( 0,  1)
0030                    ;
0031 D01D CD45D0        LOOP2:  CALL  CPHLDE           ;2 byte compare
0032 D020 3F                    CCF
0033 D021 38E0                  JR    C,RIGHT          ;if left end then go RIGHT
0034 D023 CD4BD0                CALL  UFO              ;print UFO
0035 D026 1B                    DEC   DE               ;VRAM = VRAM - 1
0036 D027 CD79D0                CALL  WAIT             ;wait
0037 D02A 18F1                  JR    LOOP2
0038                    ;
0039                    ;----- sub routines -----
0040                    ;
0041 D02C 110030        CLS:    LD    DE,3000H         ;VRAM address of ( 0,  0)
0042 D02F 21E733        LP:     LD    HL,33E7H         ;VRAM address of (39,24)
0043 D032 CD45D0                CALL  CPHLDE           ;end check
0044 D035 D8                    RET   C                ;if DE > 33E7H then return
0045 D036 3E20                  LD    A,20H            ;space code
0046 D038 42                    LD    B,D              ;BC = DE (text VRAM address)
0047 D039 4B                    LD    C,E
0048 D03A ED79                  OUT   (C),A            ;print space
0049 D03C CBA0                  RES   4,B              ;BC = attribute VRAM address
0050 D03E 3E07                  LD    A,07H            ;color 7
0051 D040 ED79                  OUT   (C),A            ;attribute out
0052 D042 13                    INC   DE               ;VRAM = VRAM + 1
0053 D043 18ED                  JR    LP               ;goto loop
0054                    ;
0055 D045 E5            CPHLDE:PUSH  HL                ;VRAM address push
0056 D046 B7                    OR    A                ;reset Cy flag
0057 D047 ED52                  SBC   HL,DE            ;if HL < DE then Cy = 1
0058 D049 E1                    POP   HL               ;VRAM address pop
0059 D04A C9                    RET
0060                    ;
0061 D04B E5            UFO:    PUSH  HL               ;VRAM address push
0062 D04C D5                    PUSH  DE
0063 D04D D5                    PUSH  DE               ;line top address push
0064 D04E DD2184D0              LD    IX,DATA          ;UFO data top address
0065 D052 0E06                  LD    C,6              ;line counter (6 lines)
0066 D054 060E          NEWLN:  LD    B,14             ;character counter of one line
0067 D056 C5            LINE:   PUSH  BC               ;counter push
0068 D057 42                    LD    B,D              ;BC = DE (text VRAM address)
```

```
0069 D058 4B                LD    C,E
0070 D059 DD7E00            LD    A,(IX+0)      ;A = UFO character data
0071 D05C ED79              OUT   (C),A         ;out to text VRAM
0072 D05E CBA0              RES   4,B           ;BC = attribute VRAM address
0073 D060 3E27              LD    A,27H         ;A = attribute code (cgen 1)
0074 D062 ED79              OUT   (C),A         ;attribute out
0075 D064 13                INC   DE            ;VRAM = VRAM + 1
0076 D065 DD23              INC   IX            ;data address increment
0077 D067 C1                POP   BC            ;counter pop
0078 D068 10EC              DJNZ  LINE          ;count dec. & repeat until count 0
0079 D06A E1        NEXTLN:POP    HL            ;HL = top address of old line
0080 D06B 112800            LD    DE,40
0081 D06E 19                ADD   HL,DE         ;HL = HL + 40
0082 D06F 54                LD    D,H           ;DE = top address of new line
0083 D070 5D                LD    E,L
0084 D071 D5                PUSH  DE            ;line top address push
0085 D072 0D                DEC   C             ;line counter decrement
0086 D073 20DF              JR    NZ,NEWLN      ;if count > 0 then go NEWLN
0087 D075 D1                POP   DE            ;dummy pop
0088 D076 D1                POP   DE            ;VRAM address pop
0089 D077 E1                POP   HL
0090 D078 C9                RET
0091                ;
0092 D079 E5        WAIT:   PUSH  HL            ;register push
0093 D07A 210008            LD    HL,800H       ;counter initialize
0094 D07D 2B        LP2:    DEC   HL            ;counter decrement
0095 D07E 7C                LD    A,H
0096 D07F B5                OR    L
0097 D080 20FB              JR    NZ,LP2        ;wait until count 0
0098 D082 E1                POP   HL            ;register back
0099 D083 C9                RET
0100                ;
0101                ;----- UFO character data -----
0102                ;
0103 D084 11213141  DATA:   DB    11H,21H,31H,41H,51H,61H,71H
     D088 516171
0104 D08B 8191A1B1          DB    81H,91H,0A1H,0B1H,0C1H,0D1H,0E1H
     D08F C1D1E1
0105 D092 12223242          DB    12H,22H,32H,42H,52H,62H,72H
     D096 526272
0106 D099 8292A2B2          DB    82H,92H,0A2H,0B2H,0C2H,0D2H,0E2H
     D09D C2D2E2
0107 D0A0 13233343          DB    13H,23H,33H,43H,53H,63H,73H
     D0A4 536373
0108 D0A7 8393A3B3          DB    83H,93H,0A3H,0B3H,0C3H,0D3H,0E3H
     D0AB C3D3E3
0109 D0AE 14243444          DB    14H,24H,34H,44H,54H,64H,74H
     D0B2 546474
0110 D0B5 8494A4B4          DB    84H,94H,0A4H,0B4H,0C4H,0D4H,0E4H
     D0B9 C4D4E4
0111 D0BC 15253545          DB    15H,25H,35H,45H,55H,65H,75H
     D0C0 556575
0112 D0C3 8595A5B5          DB    85H,95H,0A5H,0B5H,0C5H,0D5H,0E5H
     D0C7 C5D5E5
0113 D0CA 16263646          DB    16H,26H,36H,46H,56H,66H,76H
     D0CE 566676
0114 D0D1 8696A6B6          DB    86H,96H,0A6H,0B6H,0C6H,0D6H,0E6H
     D0D5 C6D6E6
0115                ;
0116 D0D8                   END
```

今度は、上段から下段までの表示時間のズレが無視できるほど小さいため先程のような歪みが消えたと思います。また全体にスピードアップしていますから、UFO が機敏に左右に移動しますね。

1-8 倍文字表示機能

アトリビュート・データの上位2ビットは、表示文字の大きさ指定に使われています。これらのビットのON／OFFは、HuBASICではCSIZEというコマンドにより行なわれ、実際に倍文字表示をする時はPRINT#0と組み合わせることになっています。

仕組みの理解も兼ねて、この部分をマシン語でプログラムしてみます。以下に掲げるサンプルプログラムは、40桁モードの画面（ページは0とする）の左下隅に、文字Aを水平垂直2倍、白色で表示するものです。今回は、BASICを「アセンブラ」風に使って、リストを表現してみました。アセンブラをお持ちでない方は、このようにすると、ハンドアセンブルに便利かと思います。

出力するASCIIコードは41Hで、文字Aのコードです。これはDレジスタに入れておきます。また、アトリビュート指定コードはC7Hで、アトリビュート・ビットのHS，VS，G，R，Bを1に、他のビットを0にしています。こうすると、HuBASICでのCSIZE 3：COLOR 7を指定したと同じことになりますね。このデータはEレジスタに入れておきます。

文字Aを指定アトリビュートでVRAMに出力するサブルーチンは、CHROUTとラベルをつけた部分で、メモリー上D100H番地から格納しました（行番号610～670)。さて、注意していただきたいのは、440行、460行、530行、550行で計4回もこのサブルーチンが呼び出されている点です。

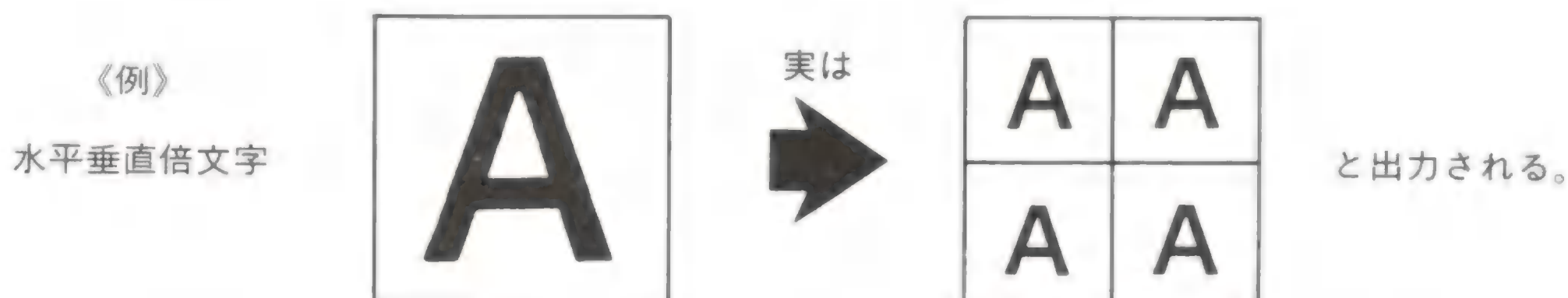
すなわち、倍文字表示の時には、単にアトリビュートを指定するだけではなく、その表示位置を同一のASCIIコードで満たさなくてはならないことです。

《例》
水平垂直倍文字

A

実は

A	A
A	A

と出力される。

これはテキスト画面への表示の段階で「見かけ上の」倍文字を実現しているからですね。マニュアルの「CSIZE」の説明中に細かい注意が書かれているのもそのためですし、プリンタでハード・コピーをとると「真の姿」が現われてしまいます。

リスト　　倍文字表示

```
100 REM **************************
110 REM *                        *
120 REM *     ﾊﾞｲﾓｼﾞ ﾋｮｳｼﾞ       *
130 REM *                        *
140 REM *   by  Y.Shimizu        *
150 REM *                        *
160 REM *      1984.7.15         *
170 REM *                        *
180 REM **************************
190 '
200 CLEAR &HD000
210 WIDTH 40 : INIT : CLS
220 ADR=&HD000 : RESTORE 420 : GOSUB "ｶｷｺﾐ" :REM ﾒｲﾝ
230 ADR=&HD100 : RESTORE 610 : GOSUB "ｶｷｺﾐ" :REM ｻﾌﾞ
240 '
250 CALL &HD000
260 '
```

```
270 END
280 '
290 LABEL "ｶｷｺﾐ" :REM ----------------------------------------------------
300 '
310 READ MC$
320 IF MC$="END" THEN RETURN
330 POKE ADR,VAL("&H"+MC$)
340 ADR=ADR+1
350 GOTO 310
360 RETURN
370 '
380 REM -- ﾏｼﾝｺﾞ ﾙｰﾁﾝ --------------------------------------------------
390 '
400                   :REM           ORG  0D000H
410                   :REM  ;
420 DATA 11,C7,41     :REM  CSIZE:   LD   DE,41C7H    ;D=ASCII , E=attribute
430 DATA 01,98,33     :REM           LD   BC,3398H    ;LOCATE 0,23
440 DATA CD,00,D1     :REM           CALL CHROUT      ;PRINT
450 DATA 03           :REM           INC  BC          ;LOCATE 1,23
460 DATA CD,00,D1     :REM           CALL CHROUT      ;PRINT
470 DATA 60           :REM           LD   H,B         ;HL=BC
480 DATA 69           :REM           LD   L,C
490 DATA 01,27,00     :REM           LD   BC,39
500 DATA 09           :REM           ADD  HL,BC       ;HL=HL+39
510 DATA 44           :REM           LD   B,H         ;LOCATE 0,24
520 DATA 4D           :REM           LD   C,L
530 DATA CD,00,D1     :REM           CALL CHROUT      ;PRINT
540 DATA 03           :REM           INC  BC          ;LOCATE 1,24
550 DATA CD,00,D1     :REM           CALL CHROUT      ;PRINT
560 DATA C9           :REM  EXIT:    RET
570 DATA END,         :REM  end mark (1)
580                   :REM  ;
590                   :REM           ORG  0D100H
600                   :REM  ;
610 DATA 7A           :REM  CHROUT:  LD   A,D         ;A=ASCII
620 DATA ED,79        :REM           OUT  (C),A       ;out to VRAM I/O(BC)
630 DATA CB,A0        :REM           RES  4,B         ;BC=attribute VRAM
640 DATA 7B           :REM           LD   A,E         ;A=attribute code
650 DATA ED,79        :REM           OUT  (C),A       ;attribute out
660 DATA CB,E0        :REM           SET  4,B         ;BC=text VRAM
670 DATA C9           :REM           RET
680 DATA END,         :REM  end mark (2)
690 '
700 REM -------------------------------------------------------------
```

サンプルプログラムの420行を

DATA　11, 87, 41　にすると水平2倍に、

DATA　11, 47, 41　にすると垂直2倍に、

変化しますから試してみて下さい。

1－9　画面表示のハイテクニックをめざして

以上で、マシン語によるテキスト画面表示の基本テクニックはおわかりいただけたと思います。本節からは、もう一歩奥に踏み込んでいきましょう。

たとえば、HuBASICで私たちが事もなげに行なっているWIDTHによる表示桁数の切り替えは、マシン語レベルでどのように行なえばよいのでしょう？　また、ユーザー定義キャラクタの設定を行なうDEFCHR$は、どんな仕組みで実行されるのでしょう？本節から扱う問題は、こうした画面表示の「原理」に関係するものです。話はグッと「ハードウェア」に傾いてきますし、難しくなると思います。けれども同時に、「この手でX1を操作している！」という実感も感じられる部分ですので、皆さんも元気を出してアタックしてみて下さい！

1-10　画面表示の原理

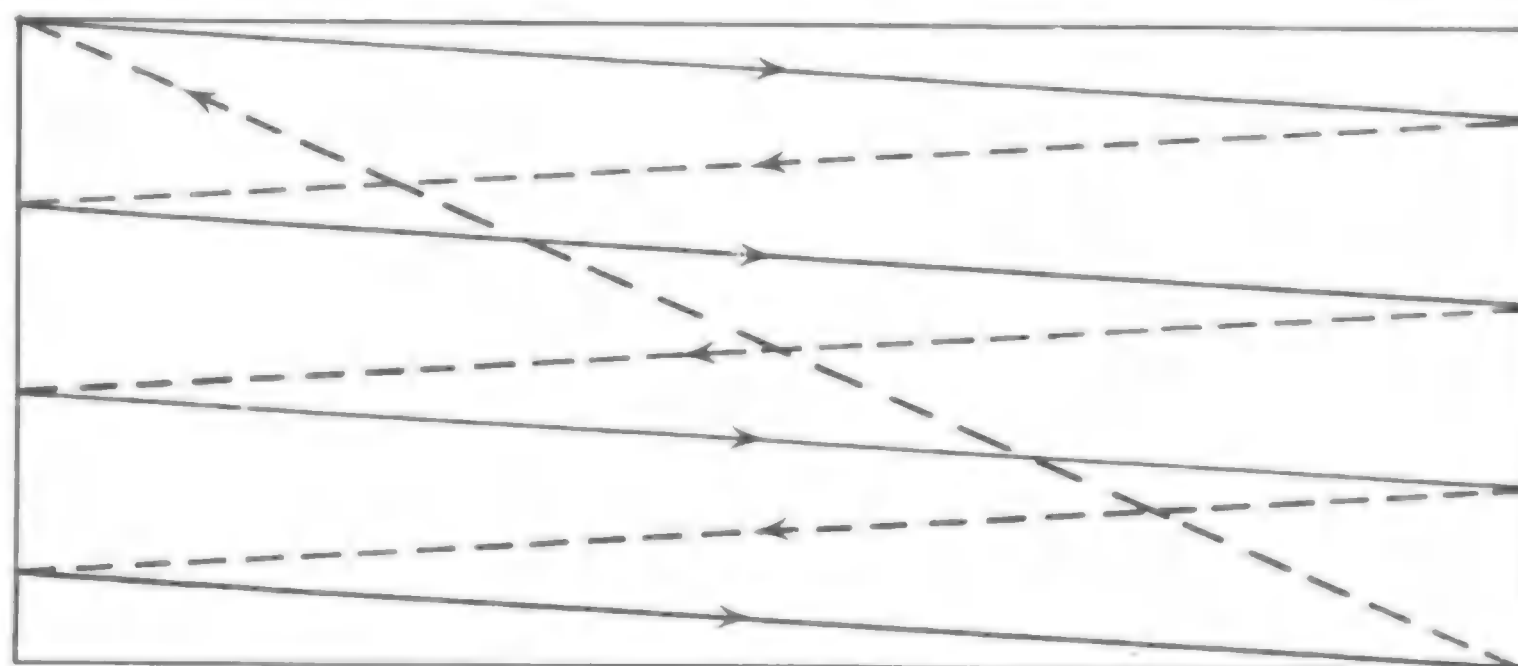
お気づきの方も多いと思いますが、コンピュータ関係では「テレビ」という言葉は余り用いられずに、**ディスプレイ、モニター、CRT** などの呼称が多く使われます。

CRT とは、Cathode Ray Tube（陰極線管と訳される）の略称で、いわゆる「ブラウン管」のことです。コンピュータでは、「ブラウン管」をテレビ放送の受像機にではなく、コンピュータからの出力装置として通常使いますから、機能を強調したモニターやディスプレイという名称や、機械の仕組みを強調した CRT という語が使われるようです。X 1 シリーズは、コンピュータとテレビの垣根を越えた画期的なマシンですから、困ってしまうのですが、以後場面に応じて、適当に使い分けていきますので、そのつもりでお読み下さい。本節以降の本章の内容では、CRT という語がよく出てくるはずです。

さて、前節で挙げたような問題を扱うには、CRT 画面にパターンを表示する原理に戻って考えてゆく必要があります。

CRT に表示される画像は、実は、電子ビームがものすごい速さで左から右へ流れて移動することにより作られています。この電子ビームの流れを**走査（scanning）**といい、流れてできる線を**走査線（ラスタ）**とよびます。よくテレビをテレビで映すと画面に縞か入りますが、あれは両者の画像の走査線のズレから生ずる現象です。

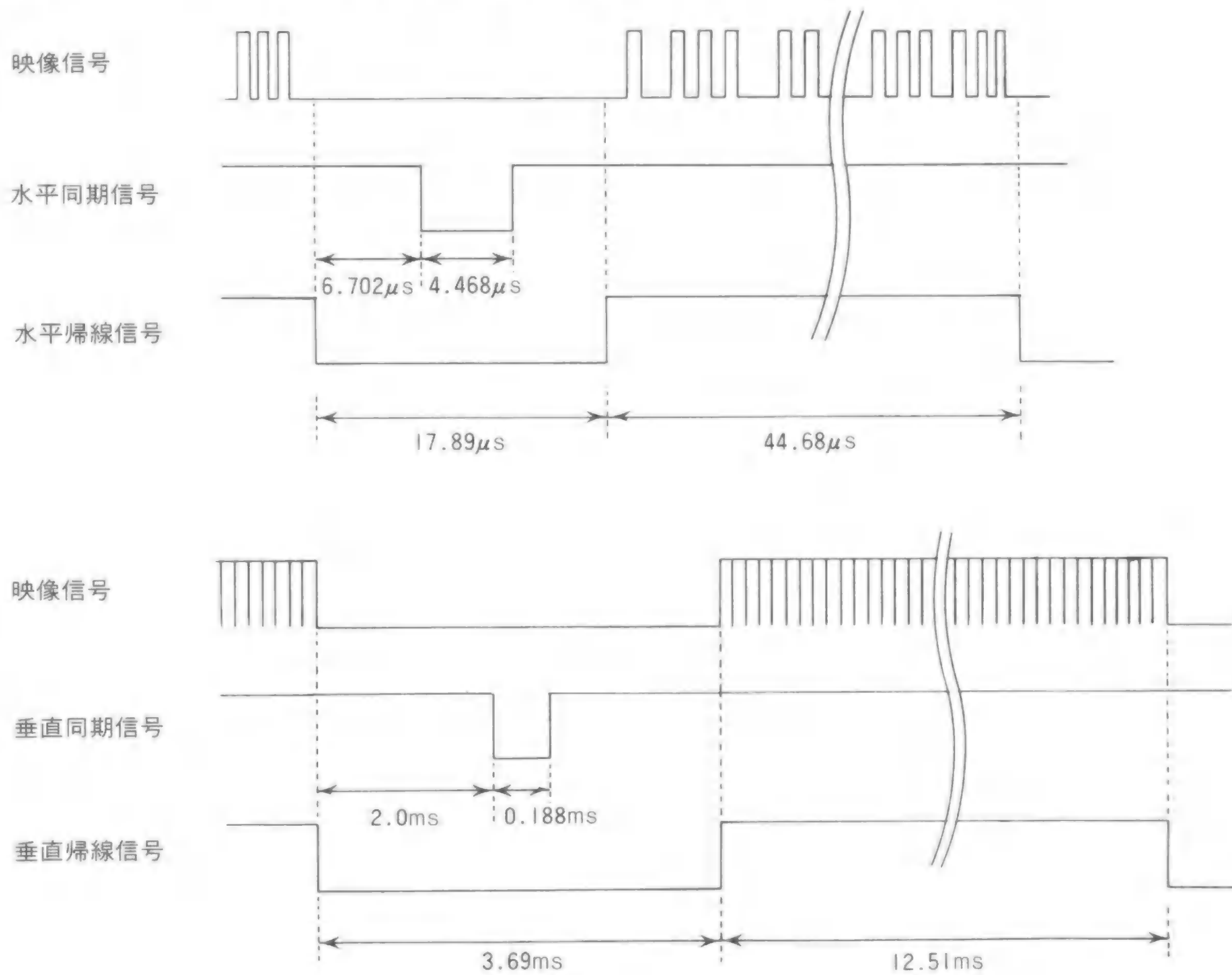
走査線の状況は次のようになっています。

走査

走査線が画面の左から右へ行きつくと、**水平帰線信号**が出され、走査線は急いで左端に戻ります（図で点線の部分。この間を**水平帰線期間**といいます）。このような繰り返しで、上から下へ走査線が進み、下端に来ると、今度は**垂直帰線信号**が発せられ、走査線は対角線を左上隅まで戻って（この間を**垂直帰線期間**といいます）、また上から走査が繰り返されます。以上の過程は一定の間隔で規則正しく行なわれます。水平帰線のタイミングをとる信号を**水平同期信号**、垂直帰線のタイミングをとる信号を**垂直同期信号**とよびます。

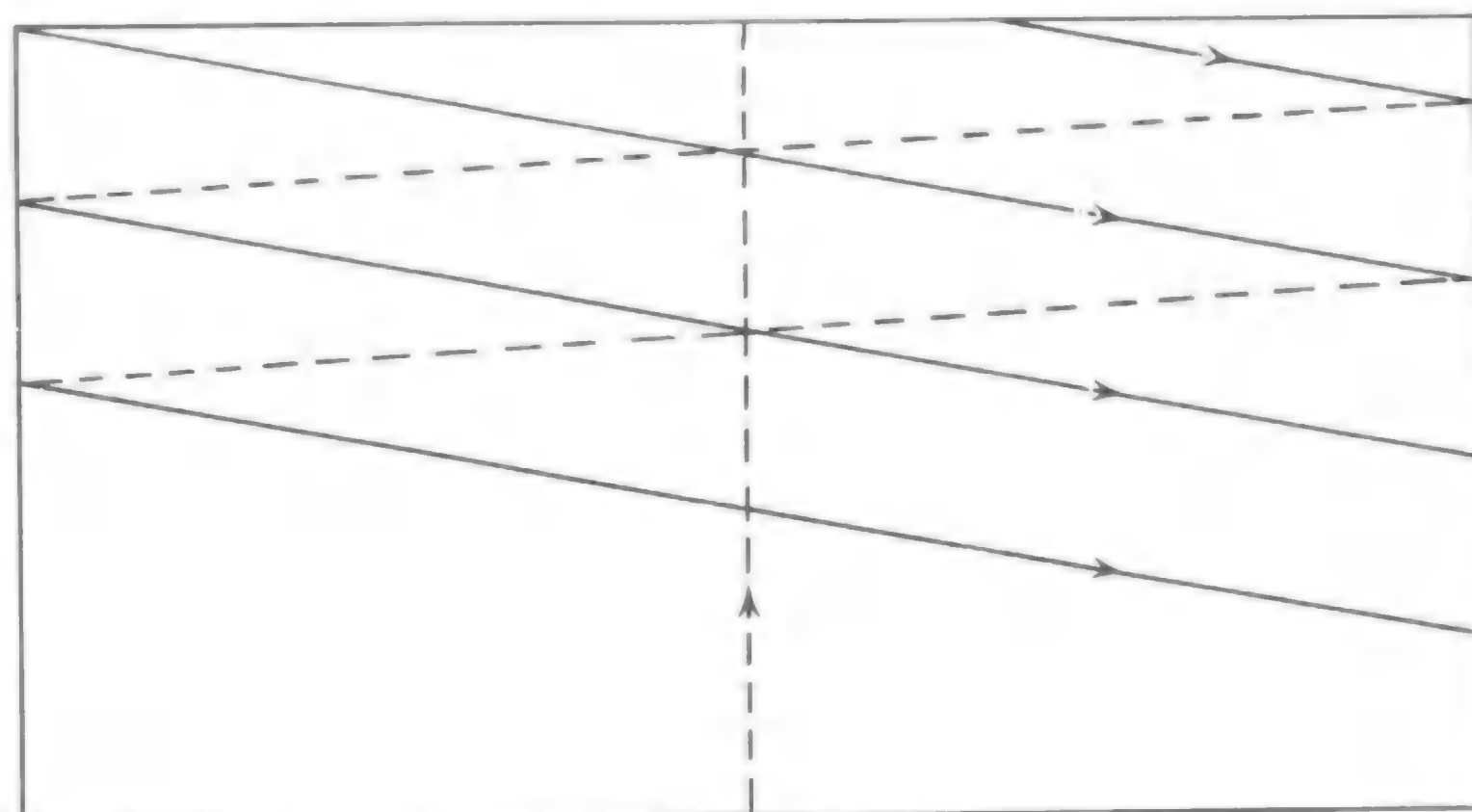
たとえば、X 1 シリーズ最初の専用ディスプレイ・テレビ（CZ-800 D）では、これらの同期信号は次のようなタイミングで発せられます（取扱説明書参照）。

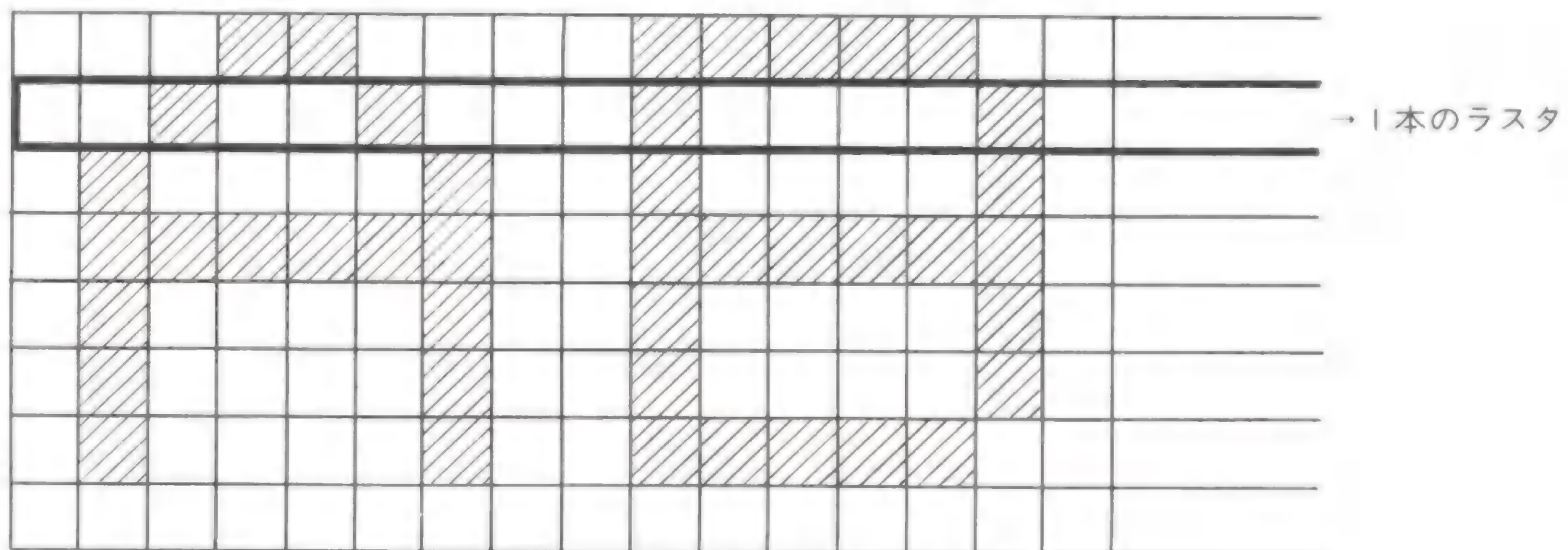
ディスプレイ・テレビの同期信号（CZ-800D）

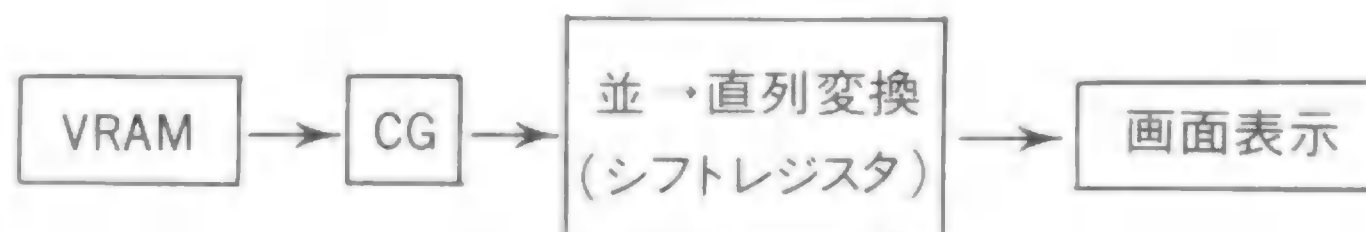
映像信号
水平同期信号
6.702μs 4.468μs
水平帰線信号
17.89μs 44.68μs

映像信号
垂直同期信号
2.0ms 0.188ms
垂直帰線信号
3.69ms 12.51ms

水平同期信号は間隔（＝信号から信号までの時間）が短く、μs（マイクロ秒＝10^{-6}秒）の単位、垂直同期信号は間隔が長く、ms（ミリ秒＝10^{-3}秒）の単位で発せられています。

［注］　家庭用テレビでは、画像の解像度をよくするために走査線の数を倍にした特殊な走査を行なっています。図のように、1つずつ飛び越して走査していく方式で、これを**インタレース方式（飛び越し走査方式）**とよびます。これに対して、前述の普通の走査方式を**順次走査**とよびます。X1のコンピュータ画面では、順次走査方式をとっているようです。しかし、テレビ画面とのスーパーインポーズ時には、特殊なことをしているはずですが、このへんは難しくて私にはよくわかりません。

飛び越し走査

以上の大枠をつかむと、CRTへの文字の表示法も想像がつくと思います。画面は文字単位というよりは、むしろ、文字の横並びをラスタ（走査線）でスライスした単位から構成され、次のラスタで文字の次のスライス部分が表示されます。

→1本のラスタ

では、AやBといった文字のパターンはどこから作り出しているのでしょうか？　これは、**キャラクタ・ジェネレータ**（Character Generator 略してCGとよぶ）とよばれるメモリー（ROM）に格納されているのです。たとえば、テキストVRAMにキャラクタ・コード41Hを格納しておくと、CGがアクセスされて、文字"A"のドット・パターンがラスタ方向にスライスされて読み出され、走査線に乗せられて画面表示される仕組みです。

VRAM → CG → 並・直列変換（シフトレジスタ） → 画面表示

X1シリーズでは、CGがROMの他にRAMも用意されている点が特徴で、RAM CGはProgrammable Character Generator（PCG）とよばれます。どちらのCGを使うかのスイッチ切り替えが、前に説明したアトリビュート指定で行なわれる訳ですね。CGからのデータ読み出しの問題は、もう少し先に扱うことにして、画面表示の原理の話を続けます。

ここまでの話からわかるように、CRT画面への文字の表示1つをとっても複雑な手続きを踏んでいる訳で、これらを全部私たちユーザーが面倒を見なければならないとしたら大変な手間です。ところが、ここに強い味方がいます。このような複雑な仕事を一手に引き受けてくれる専用LSIがあるのです。画面表示に関係する各周辺装置・回路を制御する機能をもったLSIは、**CRT controller**（**CRTコントローラ**略して**CRTC**）とよばれます。X1における画面表示のハイテクニックの多くは、CRTCの使い方に関係します。次節においてこの問題を考えましょう。

1-11　CRTコントローラ

パソコンテレビX 1では、**CRT コントローラ**(以下 CRTC と略します)として、日立製の**HD46505SP-2** というLSIを使用しています。このLSIについての詳細をお知りになりたい方は、参考文献［11］,［13］などを参照していただくことにして、ここでは必要な範囲で解説します。

このCRTCは、19個の内部レジスタを持っていて、ユーザーがこれらのレジスタに値を設定することで、必要な動作をさせることができる、いわゆる**プログラマブル**なLSIの1つです。

以下に、CRTCの内部レジスタ一覧表を掲げます。

(CRTC 内部レジスター覧表)

レジスタ記号	レジスタ名称	データビット							
		7	6	5	4	3	2	1	0
AR	アドレス・レジスタ								
R 0	水平総文字数　(注)								
R 1	水平表示文字数								
R 2	水平同期位置　(注)								
R 3	同期パルス幅	WV_3	WV_2	WV_1	WV_0	WH_3	WH_2	WH_1	WH_0
R 4	垂直総文字数　(注)								
R 5	トータル・ラスタ・アジャスト								
R 6	垂直表示文字数								
R 7	垂直同期位置　(注)								
R 8	インタレース・モード							V	S
R 9	最大ラスタ・アドレス								
R10	カーソル・スタート・ラスタ		B	P					
R11	カーソル・エンド・ラスタ								
R12	スタート・アドレス(上位)								
R13	スタート・アドレス(下位)								
R14	カーソル(上位)								
R15	カーソル(下位)								
R16	ライトペン(上位)➡X 1では未使用。								
R17	ライトペン(下位)➡X 1では未使用。								

(注)　これらのレジスタには、指定したい値から1を引いた値をセットする。

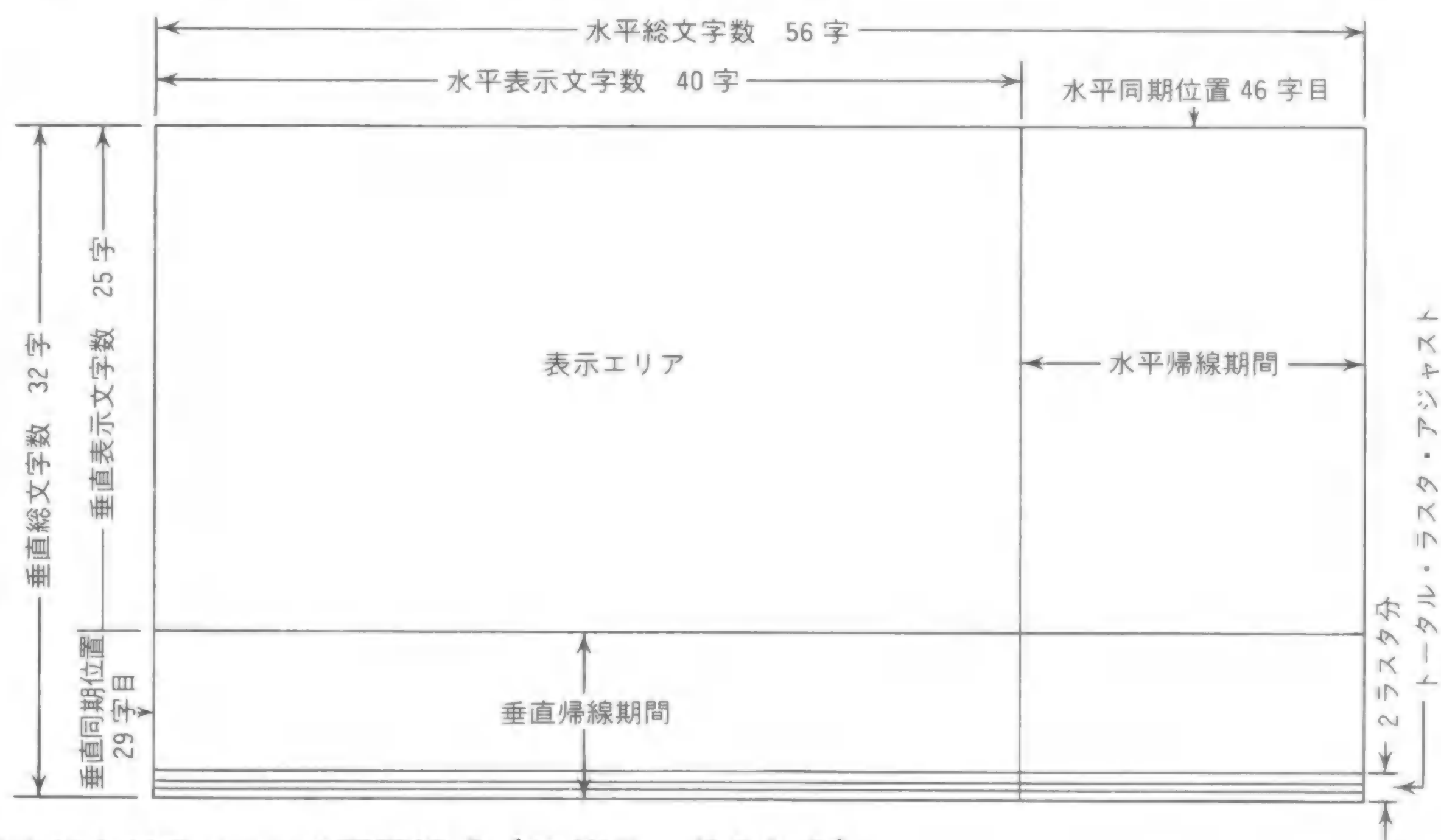
X 1においてCRTCの内部レジスタに値をセットする（プログラミング）には次の手順をとります。

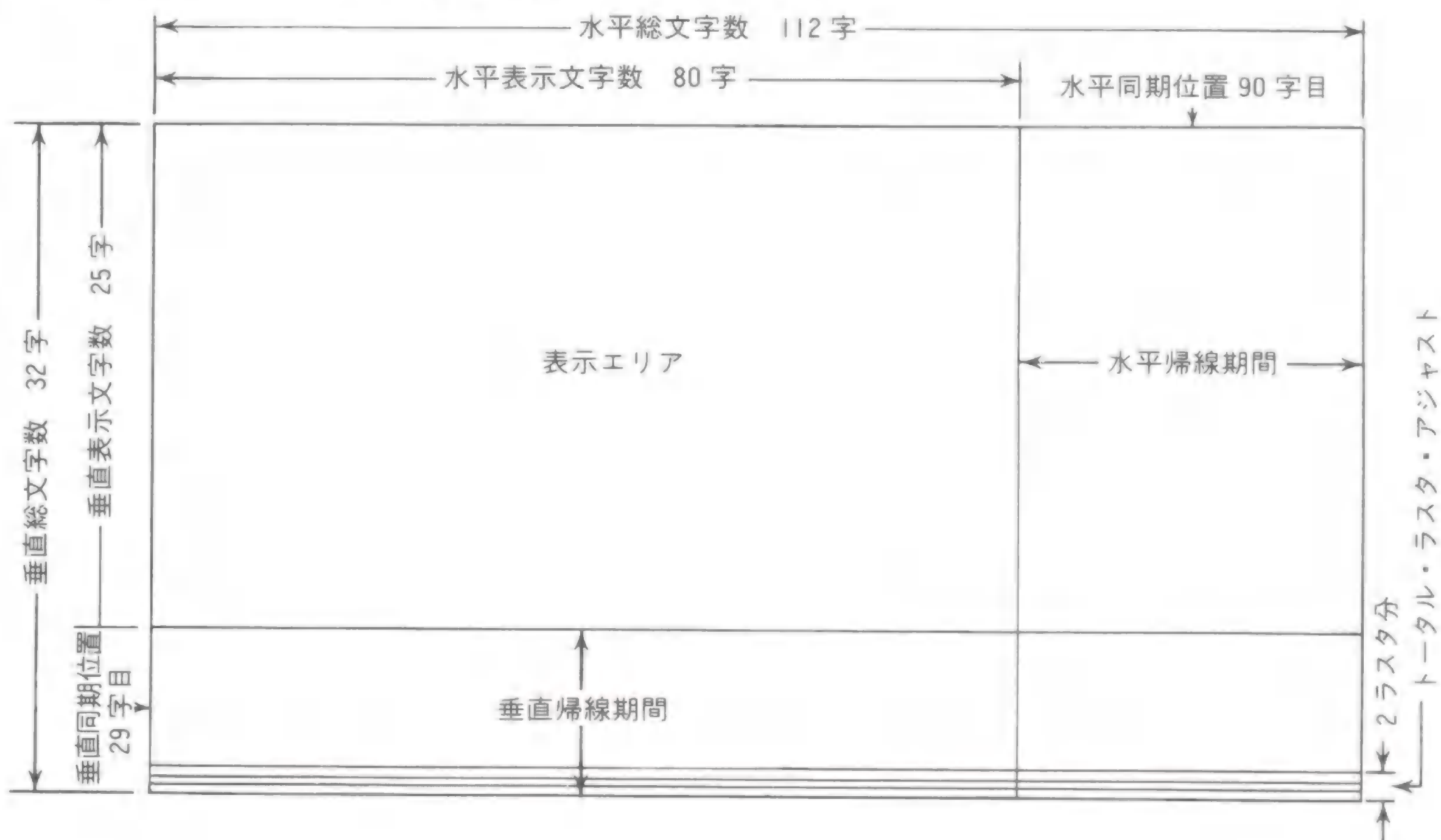
> 操作 1 ：I/Oポート1800H番地（内部レジスタAR）に、レジスタ番号（00H～0FH）を出力。
> 操作 2 ：I/Oポート1801H番地（指定したレジスタR 0 ～R15のうちの 1 つ）にデータを出力。

（注）CRTC関係のポート・アドレスは、上位 8 ビットが 18 Hであれば、最下位ビットのみ有効。たとえば、ポートアドレス 1802 Hは、1800 Hと同等、1803 Hは 1801 Hと同等です。

X 1のHuBASICでは、CRTの画面構成を次のように設定して、CRTCのプログラミングを行なっています。

X 1におけるCRTの画面構成（40 桁モードのとき）

水平総文字数　56 字
水平表示文字数　40 字
水平同期位置 46 字目
垂直総文字数　32 字
垂直表示文字数　25 字
表示エリア
水平帰線期間
垂直帰線期間
垂直同期位置 29 字目
2 ラスタ分
トータル・ラスタ・アジャスト

X 1におけるCRTの画面構成（80 桁モードのとき）

水平総文字数　112 字
水平表示文字数　80 字
水平同期位置 90 字目
垂直総文字数　32 字
垂直表示文字数　25 字
表示エリア
水平帰線期間
垂直帰線期間
垂直同期位置 29 字目
2 ラスタ分
トータル・ラスタ・アジャスト

内部レジスタＲ０，Ｒ１，Ｒ２は水平方向のデータに関係します。

Ｒ０には、画面に表示されない部分も含め、水平方向の掃引１サイクルの時間を文字数に換算し、その値から１を差し引いた値がセットされます。Ｘ１では、たとえば40桁モードなら、水平総文字数として56字分が想定されますから、Ｒ０には56－１＝55（16進数で37Ｈ）が設定されています。

Ｒ１には、画面１行あたりの実際に表示する文字数をセットします。40桁モードなら、40（16進数で28Ｈ）が設定されます。

Ｒ２には、水平同期信号をいつ発するかの位置が、文字数カウントに換算されてセットされます。実際にセットされる値は、それから１を差し引いた値です。Ｘ１では、たとえば40桁モードの時に、水平同期位置は46字目に設定されます。従って、Ｒ２には、46－１＝45(16進数で２DH)をセットすればよいのです。これを基準値より小さくすると、表示エリアの右端と水平同期位置の間隔が小さくなり、表示画面は右へズレます。逆に基準値より大きくすると表示画面は左へズレます。この様子を次の簡単なBASICプログラムで確認してみるとよいでしょう(見やすいように、40個の数字を表示して、ズレを測ります)。

リスト　　（CRTC Ｒ2）

```
100 REM ********************
110 REM *                  *
120 REM *     CRTC R2      *
130 REM *                  *
140 REM *  by Y.Shimizu    *
150 REM *                  *
160 REM *     1984.3.2     *
170 REM *                  *
180 REM ********************
190 '
200 WIDTH 40 : INIT : CLS
210 PRINT "CRTC -- ｽｲﾍｲ ﾄﾞｳｷ ｲﾁ --"
220 PRINT
230 '
240 INPUT "R2=";R2
250 OUT &H1800,2 : OUT &H1801,R2
260 FOR I=0 TO 39
270   PRINT CHR$((&H30+(I MOD 10)));
280 NEXT
290 PRINT : GOTO240
```

```
CRTC -- ｽｲﾍｲ ﾄﾞｳｷ ｲﾁ --

R2=? 1
0123456789012345678901234567890123456789

R2=? 20
0123456789012345678901234567890123456789

R2=? 45
0123456789012345678901234567890123456789

R2=? ■
```

Ｒ４，Ｒ６，Ｒ７は垂直方向に関して、各々Ｒ０，Ｒ１，Ｒ２に対応する機能を持っています。

Ｒ３は、水平、垂直の各同期信号のパルス幅をセットするレジスタです。上位４ビット（WV３～WV０）で垂直同期パルス幅をラスタ本数に換算した値を、下位４ビット（WH３～WH０）で水平同期パルス幅を文字数に換算した値を指定します。Ｘ１では、垂直同期パルス幅は３ラスタ分、水平同期パルス幅は、40桁モードで４文字分、80桁モードで８文字分となるようにセットされます。従って、40桁モードの時、Ｒ３にセットされる値は34Ｈになります。

R 5の役割について説明します。垂直方向の掃引周期は、1文字あたりのラスタ数8と、垂直総文字数との積で決まってしまいますが、これではディスプレイ・テレビの仕様とズレが生じるかもしれません。そのための微調整（トータル・ラスタ・アジャスト）を行なうためのレジスタがR 5です。CRTCは、R 4による時間調整の後に、R 5で指定されたラスタ本数の分だけ余分に時間かせぎをします。X 1では、R 5は値2にセットされています（ディスプレイ CZ-800 Dの場合)。

R 8は、画面を順次走査するか（ノン・インタレース)、飛び越し走査するか（インタレース）等を指定するレジスタです。X 1では、ノン・インタレース・モードを採用し、R 8には、00 Hをセットします。

R 9には、1行に対応するラスタ数から1を差し引いた値（最大ラスタ・アドレス）がセットされます。X 1では、1行は8本のラスタから構成され、0～7のラスタ・アドレスを持ちますから、R 9には07 Hをセットします。

R 10とR 11はカーソル制御のためのレジスタですが、X 1では豊富なアトリビュート機能を用いて、ソフトウェア的にカーソルを表示するために、CRTCのカーソル機能は用いません。一応、R 10には値60 Hが、R 11には値07 Hがセットされますが、別の値にセットし直しても変化は見られません。

R 12とR 13は、画面のホーム位置（x座標0，y座標0）に対応するVRAMの読み出し先頭アドレスをセットするレジスタです。ただし、VRAM先頭アドレスを0000 Hとする相対アドレスで計算されます。X 1では、通常0000 Hにセットされていますが、40桁モード SCREEN 1 のときは、テキストVRAMは3400 H番地からですから、0400 H（R 12=04 H，R 13=00 H)にセットされます。従って、**画面のページ切り替え**をマシン語で行なう時は、ここを操作します。

R16とR17は，ライトペン制御用のレジスタですが，X1ではこの機能を用いていないので省略します。R14 と R15 は未使用なので，値0 0 H にセットします。

1-12 表示桁数の切り替え

CRTCによる40桁モード、80桁モードの画面構成を理解すると、WIDTH 40, WIDTH 80 に相当することをマシン語で実現することができます。

まず、HuBASICのシステム・サブルーチンを用いた方法を紹介します（コール・アドレスは、テープBASICでも、ディスクBASICでも同じです)。

表示文字切り替えのシステム・サブルーチン

表示文字数切り替えのシステム・サブルーチン	
WIDTH 40：0 9 9 8 H	Hu BASIC内サブルーチン・コール・アドレス
WIDTH 80：0 9 8 C H	

BASICのCALL命令で、これらのサブルーチンを呼び出し、表示文字数が切り替わることを確認して下さい。（例：CALL &H98Cで80桁モードになります。）

次に表示文字数切り替えの原理を知るために、主要部分をマシン語プログラムとして作ってみましょう。

第一に、40桁モード、80桁モードの切り替えスイッチのセットが必要です。そのためには、I／Oポートの1A03H番地を用います。

表示文字数切り替えスイッチ

表示文字数切り替えスイッチ
40桁モード：I/OポートＩＡ０３H番地へ０DHを出力。
80桁モード：I/OポートＩＡ０３H番地へ０CHを出力。

（注）詳細は第4章で述べますが、Ｉ／ＯポートＩＡ03Ｈ番地は、インターフェースLSI 8255のポートで、ここに０DHを出力すると、8255Cポートのビット６がセットされ、0CHを出力すると、Ｃポートのビット６がリセットされます。すなわち、8255のＣポート・ビット６が表示文字数切り替えスイッチになっているのです。

さて、これで切り替えができるのですが、このままではいけないことは前節の議論でおわかりと思います。CRTCは、40桁モード、80桁モードの各々で異なった画面構成で機能しますから、対応するデータをCRTCの内部レジスタにセットしなくてはなりません。CRTCの復習も兼ねて次のプログラムを御覧下さい。

このプログラムは、テンキーの4と8を押すことで、一瞬に40桁モード、80桁モードの切り替えを行ないます。HuBASIC内システムサブルーチンのように切り替えに伴う画面クリアを行ないませんから、面白い現象が生じます。プログラムの終了は、ESC キーを押して下さい。BASICのコマンド・レベルに復帰します。

リスト　　（CRTC control program）

```
0000 ********************************
0001 *                              *
0002 *  CRTC control   program      *
0003 *                              *
0004 *     ( Hu BASIC mode )        *
0005 *                              *
0006 *       by Y.Shimizu           *
0007 *                              *
0008 *       1984.1.7  SAT          *
0009 *                              *
0010 ********************************
0011                    ;
0012 1900               KEYBD: EQU  1900H           ;KEYBOARD DATA PORT
0013 001B               ESC:   EQU  1BH             ;ESC key code
0014 0034               TKEY4: EQU  34H             ;ten key '4' code
0015 0038               TKEY8: EQU  38H             ;ten key '8' code
0016                    ;
0017 1800               CRTCRS:EQU  1800H           ;CRTC register select port
0018 1801               CRTCDT:EQU  1801H           ;CRTC data set port
0019                    ;
0020 1A03               CT8255:EQU  1A03H           ;8255 control register port
0021 000C               WID80: EQU  0CH             ;8255 PC6 bit reset (80 mode)
0022 000D               WID40: EQU  0DH             ;8255 PC6 bit set   (40 mode)
0023                    ;
0024 000B               PRMESG:EQU  000BH           ;message print (DE=pointer,00=EOL)
0025 0000               INIT:  EQU  0000H           ;system initialize
0026                    ;
0027                           ORG  0E000H
```

```
0028 E000 3100F0          LD    SP,0F000H
0029                      ;
0030                      ;
0031                      ; ***  INPUT ROUTINE ***
0032                      ;
0033 E003 111BE0   MESSG: LD    DE,MSG1         ;prompt message print
0034 E006 CD0B00          CALL  PRMESG
0035 E009 CD3BE0   INPUT: CALL  INKEY
0036 E00C FE1B            CP    ESC
0037 E00E CA0000          JP    Z,INIT          ;IF ESC THEN INITIALIZE
0038 E011 FE34            CP    TKEY4
0039 E013 2836            JR    Z,MODE40        ;IF "4" THEN WIDTH40
0040 E015 FE38            CP    TKEY8
0041 E017 283E            JR    Z,MODE80        ;IF "8" THEN WIDTH80
0042 E019 18EE            JR    INPUT
0043                      ;
0044 E01B 0D       MSG1:  DB    0DH             ;preparation for message
0045 E01C 57494454        DEFM  'WIDTH MODE (4 or 8 or ESC) = ?'
     E020 48204D4F
     E024 44452028
     E028 34206F72
     E02C 2038206F
     E030 72204553
     E034 4329203D
     E038 203F
0046 E03A 00              DB    00H             ;End Of Line
0047                      ;
0048 E03B 010019   INKEY: LD    BC,KEYBD        ;IF "4" OR "8" OR ESC THEN RET
0049 E03E ED78            IN    A,(C)
0050 E040 FE1B            CP    ESC
0051 E042 C8              RET   Z
0052 E043 FE34            CP    TKEY4
0053 E045 C8              RET   Z
0054 E046 FE38            CP    TKEY8
0055 E048 C8              RET   Z
0056 E049 18F0            JR    INKEY
0057                      ;
0058                      ; *** MODE CHANGE ROUTINE ***
0059                      ;
0060 E04B 01031A   MODE40:LD    BC,CT8255
0061 E04E 3E0D            LD    A,WID40
0062 E050 ED79            OUT   (C),A           ;change to 40 mode
0063 E052 2179E0          LD    HL,DATA40       ;HL = data top adrs.
0064 E055 180A            JR    MAIN
0065                      ;
0066 E057 01031A   MODE80:LD    BC,CT8255
0067 E05A 3E0C            LD    A,WID80
0068 E05C ED79            OUT   (C),A           ;change to 80 mode
0069 E05E 217AE0          LD    HL,DATA80       ;HL = data top adrs.
0070                      ;
0071                      ; *** CRTC DATA SET ROUTINE ***
0072                      ;
0073 E061 1600     MAIN:  LD    D,0             ;D = CRTC register code (0-13)
0074 E063 010018   LOOP:  LD    BC,CRTCRS       ;CRTC register select
0075 E066 ED51            OUT   (C),D
0076 E068 010118          LD    BC,CRTCDT       ;CRTC data set
0077 E06B 7E              LD    A,(HL)          ;HL = data pointer
0078 E06C ED79            OUT   (C),A
0079 E06E 23              INC   HL              ;pointer increment
0080 E06F 23              INC   HL
0081 E070 14              INC   D               ;register code increment
0082 E071 7A              LD    A,D
0083 E072 FE0E            CP    14
0084 E074 38ED            JR    C,LOOP
0085                      ;
0086 E076 C309E0          JP    INPUT           ;GOTO INPUT ROUTINE
0087                      ;
0088                      ; *** OUTPUT DATA for CRTC ***
0089                      ;
0090 E079 37       DATA40:DB     55              ;ｽｲﾍｲ ｿｳﾓｼﾞｽｳ -1 (40 mode)
0091 E07A 6F      DATA80:DB     111             ;                (80 mode)
0092 E07B 28              DB    40              ;ｽｲﾍｲ ﾋｮｳｼﾞﾓｼﾞｽｳ
0093 E07C 50              DB    80
0094 E07D 2D              DB    45              ;ｽｲﾍｲ ﾄﾞｳｷ ｲﾁ -1
0095 E07E 59              DB    89
0096 E07F 34              DB    34H             ;ﾄﾞｳｷ ﾊﾟﾙｽ ﾊﾊﾞ
0097 E080 38              DB    38H
0098 E081 1F              DB    31              ;ｽｲﾁｮｸ ｿｳﾓｼﾞｽｳ -1
0099 E082 1F              DB    31
0100 E083 02              DB    2               ;ﾄｰﾀﾙ ﾗｽﾀ ｱｼﾞｬｽﾄ
0101 E084 02              DB    2
0102 E085 19              DB    25              ;ｽｲﾁｮｸ ﾋｮｳｼﾞﾓｼﾞｽｳ
0103 E086 19              DB    25
0104 E087 1C              DB    28              ;ｽｲﾁｮｸ ﾄﾞｳｷ ｲﾁ -1
0105 E088 1C              DB    28
0106 E089 00              DB    00H             ;ﾗｽﾀ ｽｷｬﾝ ﾓｰﾄﾞ (ｲﾝﾀﾚｰｽ)
0107 E08A 00              DB    00H
0108 E08B 07              DB    7               ;ｻｲﾀﾞｲ ﾗｽﾀ ｱﾄﾞﾚｽ (ﾗｽﾀ=8ﾎﾝ)
0109 E08C 07              DB    7
```

```
0110 E08D 60              DB    60H           ;ｶｰｿﾙ ｽﾀｰﾄ ﾗｽﾀ
0111 E08E 60              DB    60H
0112 E08F 07              DB    07H           ;ｶｰｿﾙ ｴﾝﾄﾞ ﾗｽﾀ
0113 E090 07              DB    07H
0114 E091 00              DB    00H           ;VRAM read top adrs. High
0115 E092 00              DB    00H
0116 E093 00              DB    00H           ;VRAM read top adrs. Low
0117 E094 00              DB    00H
0118              ;
0119 E095                 END
```

1-13 プログラマブル・キャラクタ・ジェネレータ（Ⅰ）

X 1においては、通常のROMキャラクタ・ジェネレータの他に、RAMキャラクタ・ジェネレータが装備されていて、ここにユーザーが好きなパターンをプログラムすることで、ユーザー定義文字を256文字も使うことができます。RAMキャラクタ・ジェネレータは、Programmable Character Generator ともよばれ、以下頭文字をとって PCG とよぶことにします。

PCGにパターンを定義するために、HuBASICでは、DEFCHR$命令が用意されていますが、本節では、マシン語による制御法を解説します。

PCGでは、各キャラクタ（8×8ドット）のドット毎に色を指定できるように、Blue, Red, Greenの3原色に対応するRAMが用意されています。ユーザーは各原色につき8バイト分のパターンを定義し、合計24バイトのデータをもって、1つのキャラクタ・パターンができあがります。

これらのRAMをアクセスするためには、I／Oポートの1500H～17FFH番地が割りあてられています。詳しくは、次のようになります。

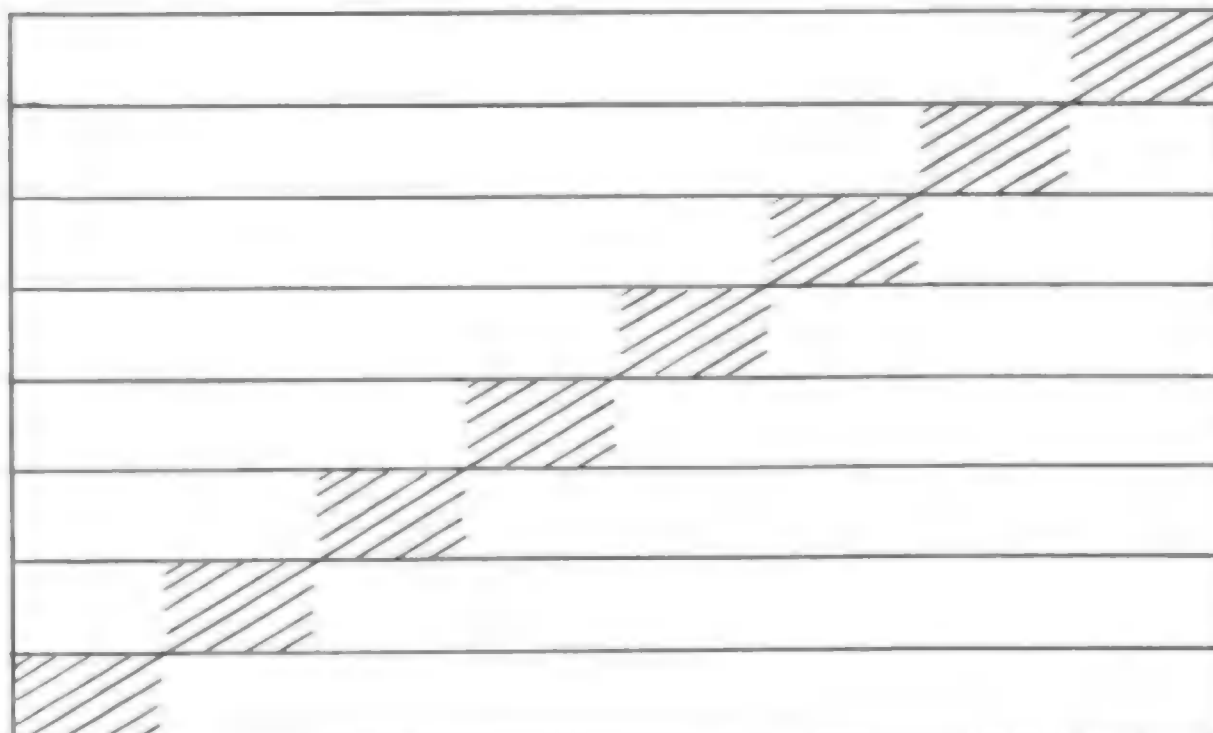
PCGアクセスのI/Oアドレス

PCG(Blue)	：I/Oポート	1500H～1507H番地
PCG(Red)	：I/Oポート	1600H～1607H番地
PCG(Green)	：I/Oポート	1700H～1707H番地

たとえば、PCG（B）を例にとると、定義したいキャラクタのパターンと、I／Oポートのアドレスの関係は次のようになっています。

PCG（B）とキャラクタ・パターン

I／Oアドレス

1500H番地
1501H番地
1502H番地
1503H番地
1504H番地
1505H番地
1506H番地
1507H番地

PCG（B）に前ページのようなパターンを定義したければ、
I／Oポートに次のようなデータを出力する。

I／O（1500H）←01H
I／O（1501H）←02H
I／O（1502H）←04H
I／O（1503H）←08H
I／O（1504H）←10H
I／O（1505H）←20H
I／O（1506H）←40H
I／O（1507H）←80H

ついでに、ROMキャラクタ・ジェネレータ（ROMCGと以下略記する）をアクセスするためのI／Oアドレスも述べおきます。

ROMCGアクセスのI/Oアドレス
ROMCG：I/Oポート　1400H番地～1407H番地

（注）これらのI／Oアドレスでは、アドレス・バスの上位8ビット（14H，15H，16H，17H）で、キャラクタ・ジェネレータの区別が行なわれ、下位3ビットで、1キャラクタの何段目（0～7）を指定するかが区別され、他のビットは無効となります。

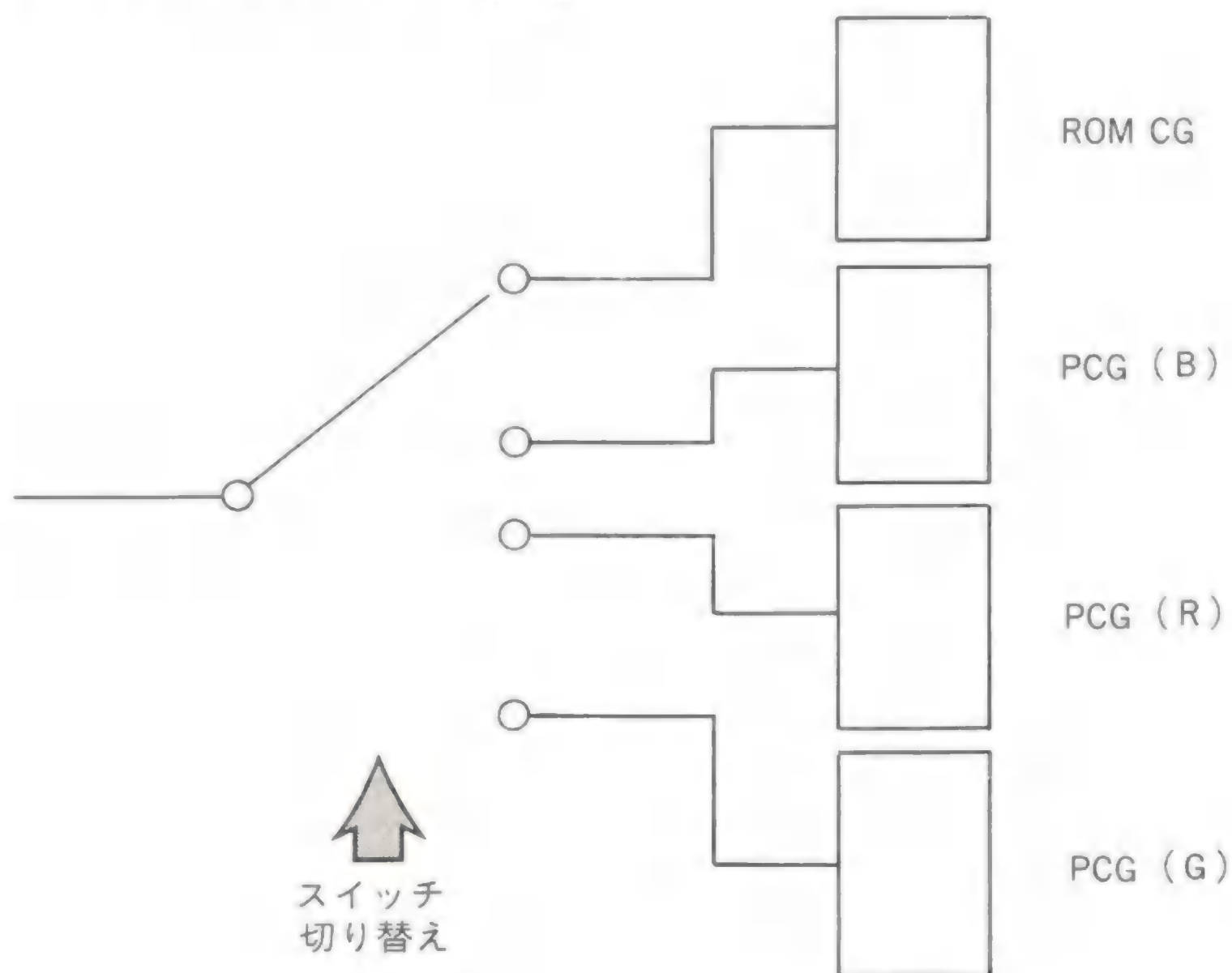
〈例〉　アドレス・バス　00010101×××××000

上位8ビット　15Hゆえ、PCG(B)を選択
無効
下位3ビット0ゆえ、0段目(上から1段目)を選択

ですから、たとえばPCG（B）において、I／Oアドレス1508H番地は1500H番地と同等になります（8の倍数のズレは同等！）。

以上がキャラクタ・ジェネレータ関係のI／Oアドレスですが、疑問を持たれた読者もおられるでしょう。キャラクタ番号(ASCIIコード)の区別はどのようにしているか？　という当然の疑問かと思われます。

キャラクタ・ジェネレータ（ROM，RAMとも）もメモリーである以上、きちんとアドレスがつけられていて、各ASCIIコードに対応するキャラクタのドット・パターンが格納されている訳ですが、X1においては必要最小限の回路構成でアクセス可能とするために、ROMCGとPCG（RAMCG）たちは、全く同じアドレス配置をとっています。

キャラクタ・ジェネレータの配置

このために、各CGは、I／O空間上に特定のエリアを持つことなく、I／Oアドレスの上位8ビットで選択されたCGの1キャラクタ分だけがアクセス可能となる構成がなされています。

さて、キャラクタ番号の指定法ですが、以上のことから、かなり特殊な方法であることは想像できるでしょう。読者の中には、テキストVRAM、アトリビュートVRAMとして割りあてられているI／Oアドレスのうち、表示に用いられるエリアは全部ではなく、余ったエリアは何のためにあるかと疑問を持たれた方もおられるでしょう。次節のテーマは、このVRAMの余ったエリアと関係します。

1-14 プログラマブル・キャラクタ・ジェネレータ（II）

各表示モードでの、VRAMの余ったエリア（表示に用いられない）は次のようになっています。

（VRAMの余ったエリア）

40桁モード、ページ0	テキストVRAM	アトリビュートVRAM
表示に使われるエリア	3000H～33E7H	2000H～23E7H
余ったエリア	33E8H～33FFH （24バイト）	23E8H～23FFH （24バイト）

40桁モード、ページ1	テキストVRAM	アトリビュートVRAM
表示に使われるエリア	3400H～37E7H	2400H～27E7H
余ったエリア	37E8H～37FFH (24バイト)	27E8H～27FFH (24バイト)

80桁モード	テキストVRAM	アトリビュートVRAM
表示に使われるエリア	3000H～37CFH	2000H～27CFH
余ったエリア	37D0H～37FFH (48バイト)	27D0H～27FFH (48バイト)

これらのエリアは、通常は、走査線が表示エリアの右下端に達した後、ホーム位置へ戻るまでの垂直帰線期間内のエリアとして用られますが、もう1つの用途として、キャラクタ・ジェネレータをアクセスするための役割を持っています。

すなわち、キャラクタ・ジェネレータを（読みとり、あるいは書き込みで）アクセスするための第1段階は次のようになっています。

CGアクセス（1）

CGは RAM, ROM？

RAMCG

ROMCG

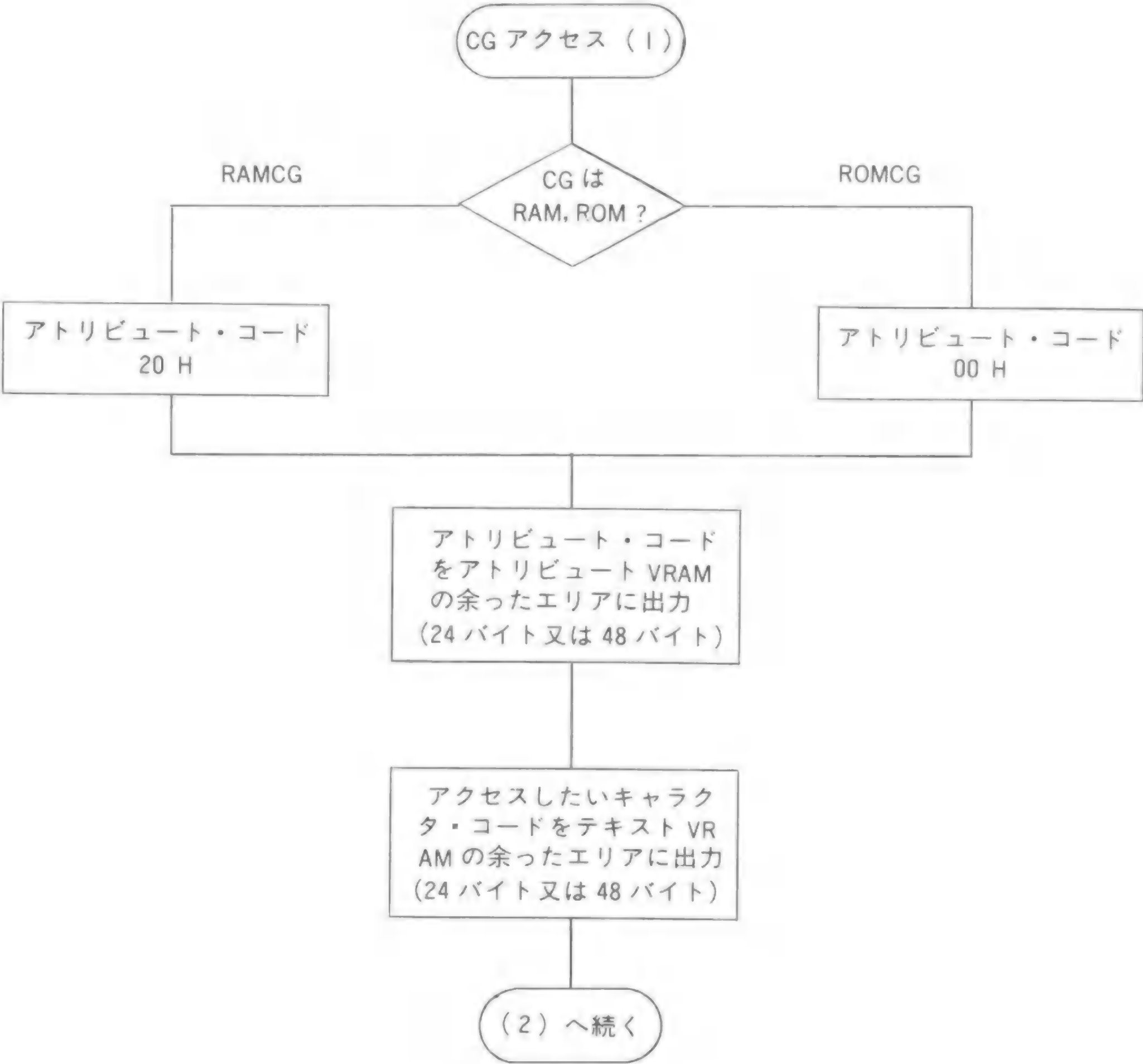
アトリビュート・コード 20 H

アトリビュート・コード 00 H

アトリビュート・コードをアトリビュートVRAMの余ったエリアに出力（24バイト又は48バイト）

アクセスしたいキャラクタ・コードをテキストVRAMの余ったエリアに出力（24バイト又は48バイト）

（2）へ続く

この段階で、読みとり、あるいは書き込みをしたいキャラクタ・コードがCG側に伝わる訳ですね。

第2段階は、タイミングをとりながら、CGとデータのやりとりをする部分です。各CGとのデータ授受は8バイトにわたって行なわれ、前節で述べたように選択するCG（PCGのB，R，GおよびROMCG）によってアクセスするI／Oアドレスは異なります。

フローチャート　CGアクセス（2）

CGとのデータ授受は、垂直帰線期間内に行なわれます。垂直帰線信号が"H"か"L"かは、I／Oポート1A01H番地のMSBで読みとれるようになっています(V-DISP信号)。

また、X 1ではキー入力を割り込み処理していて、CG アクセス時に割り込みがかかると、タイミングがズレてしまいますから、以上の過程を割り込み禁止で行なう必要があります。

さあ、これらが理解できましたら、プログラム作成をしてみましょう。簡単のために、40桁モード・ページ0の画面で、PCG (RAMCG) へのデータ書き込みのサブルーチンを作ります。後で使えるように、メモリー上FB 00 H 番地から格納しました。

リスト PCG data set subroutine

```
0000                 ;***************************************
0001                 ;*                                     *
0002                 ;*      PCG data set subroutine        *
0003                 ;*                                     *
0004                 ;***************************************
0005                 ;
0006                 ;-------------------------------------------------
0007                 ;screen mode     = WIDTH 40: SCREEN 0,0
0008                 ;
0009                 ;input reg : D   = character code
0010                 ;            E   = PCG select (15H or 16H or 17H)
0011                 ;            HL  = top address of data (8 bytes)
0012                 ;
0013                 ;output reg: HL = HL + 8 (next data address)
0014                 ;
0015                 ;preserved:  BC,DE
0016                 ;-------------------------------------------------
0017                 ;
0018                 ;
0019                         ORG   0FB00H
0020                 ;
0021                 ;
0022                 ;----- register push -----
0023                 ;
0024 FB00 C5                 PUSH BC             ;Keep BC,DE
0025 FB01 D5                 PUSH DE
0026 FB02 E5                 PUSH HL             ;data address push
0027                 ;
0028                 ;
0029                 ;----- RAM CG mode -----
0030                 ;
0031 FB03 0618       RAMCG:  LD   B,24           ;left bytes of VRAM
0032 FB05 21E823             LD   HL,23E8H       ;top address of left area of VRAM
0033 FB08 C5         LOOP1:  PUSH BC             ;counter push
0034 FB09 44                 LD   B,H            ;BC = VRAM address
0035 FB0A 4D                 LD   C,L
0036 FB0B 3E20               LD   A,20H          ;A = attribute (CGEN 1)
0037 FB0D ED79               OUT  (C),A          ;attribute out
0038 FB0F 23                 INC  HL             ;increment VRAM address
0039 FB10 C1                 POP  BC             ;counter pop
0040 FB11 10F5               DJNZ LOOP1          ;loop by count B reg
0041                 ;
0042                 ;
0043                 ;----- character code set -----
0044                 ;
0045 FB13 0618       CODSET:LD    B,24           ;left bytes of VRAM
0046 FB15 21E833             LD   HL,33E8H       ;top address of left area of VRAM
0047 FB18 C5         LOOP2:  PUSH BC             ;counter push
0048 FB19 44                 LD   B,H            ;BC = VRAM address
0049 FB1A 4D                 LD   C,L
0050 FB1B 7A                 LD   A,D            ;A = character code for PCG set
0051 FB1C ED79               OUT  (C),A          ;character code out
0052 FB1E 23                 INC  HL             ;increment VRAM address
0053 FB1F C1                 POP  BC             ;counter pop
0054 FB20 10F6               DJNZ LOOP2          ;loop by count B reg
0055                 ;
0056                 ;
0057                 ;----- preparation for PCG set -----
0058                 ;
0059 FB22 43         PREP1:  LD   B,E            ;BC = I/O address of PCG select
0060 FB23 0E00               LD   C,0
0061 FB25 E1                 POP  HL             ;HL = top address of PCG data area
0062 FB26 1E08               LD   E,8            ;E  = PCG set counter (8 bytes)
0063 FB28 C5                 PUSH BC             ;I/O address push
0064                 ;
0065 FB29 01011A     PREP2:  LD   BC,1A01H       ;V-DISP = MSB of 8255 port B
0066 FB2C ED78       WAIT2:  IN   A,(C)
0067 FB2E F22CFB             JP   P,WAIT2        ;wait until V-DISP = 1
```

```
0068                    ;
0069 FB31 F3            PREP3: DI                       ;disable interrupt
0070                    ;
0071 FB32 ED78          PREP4: IN      A,(C)
0072 FB34 FA32FB               JP      M,PREP4          ;wait until V-DISP = 0
0073                    ;
0074 FB37 C1            PREP5: POP     BC               ;BC = I/O address
0075                    ;
0076                    ;
0077                    ;----- PCG set (8 bytes) -----
0078                    ;
0079 FB38 7E            LOOP3: LD      A,(HL)           ;A = PCG data
0080 FB39 ED79                 OUT     (C),A            ;out to I/O(BC)
0081 FB3B 23                   INC     HL               ;increment data address
0082 FB3C 03                   INC     BC               ;increment I/O address
0083                    ;
0084 FB3D 00            WAIT3: NOP                      ;waiting
0085 FB3E 3E0E                 LD      A,14
0086 FB40 3D            LP:    DEC     A
0087 FB41 C240FB               JP      NZ,LP
0088                    ;
0089 FB44 1D            COUNT: DEC     E                ;byte counter decrement
0090 FB45 C238FB               JP      NZ,LOOP3         ;repeat for 8 bytes
0091                    ;
0092                    ;
0093                    ;----- end of subroutine -----
0094                    ;
0095 FB48 D1            EXIT:  POP     DE               ;registers back
0096 FB49 C1                   POP     BC
0097 FB4A FB                   EI                       ;enable interrupt
0098 FB4B C9                   RET                      ;return
0099                    ;
0100                    ;
0101 FB4C                      END
```

さて、このサブルーチンをテストするためのメインルーチンを作ります。キャラクタ・コード 30 H～33 Hに 4 種類の色で斜線／を定義してみました。

リスト　main routine for PCG set (example)

```
0000                    ;*********************************************************
0001                    ;*                                                       *
0002                    ;*      main routine for PCG set (example)               *
0003                    ;*                                                       *
0004                    ;*                  by Y.Shimizu                         *
0005                    ;*                                                       *
0006                    ;*                   1984.3.5                            *
0007                    ;*                                                       *
0008                    ;*********************************************************
0009                    ;
0010                    ;----------------------------------------
0011                    ;
0012                    ;      TEST pattern for PCG setting
0013                    ;
0014                    ;ACSII code 30H = pattern of "/" (white)
0015                    ;           31H =            "/" (blue)
0016                    ;           32H =            "/" (red)
0017                    ;           33H =            "/" (green)
0018                    ;
0019                    ;----------------------------------------
0020                    ;
0021                    ;
0022 FB00               PCGSUB:EQU  0FB00H          ;PCG data set subroutine
0023 1000               MONHOT:EQU  1000H           ;Hu BASIC monitor hot start
0024                    ;
0025                    ;
0026                           ORG  0F000H
0027                    ;
0028                    ;
0029                    ;----- initial setting -----
0030                    ;
0031 F000 1630                 LD   D,30H           ;D  = start ASCII code for PCG set
0032 F002 1E15                 LD   E,15H           ;E  = PCG select code (Blue)
0033 F004 211CF0               LD   HL,DATA         ;HL = top address of PCG data
0034 F007 0604                 LD   B,4             ;B  = counter of PCG set (4 char)
0035                    ;
0036                    ;
0037                    ;----- main loop for PCG set -----
0038                    ;
0039 F009 D5            LOOP:  PUSH DE              ;keep D,E
0040                    ;
0041 F00A CD00FB               CALL PCGSUB          ;blue pattern set
```

```
0042 F00D 1C                INC  E              ;PCG select code = 16H
0043 F00E CD00FB            CALL PCGSUB         ;red pattern set
0044 F011 1C                INC  E              ;PCG select code = 17H
0045 F012 CD00FB            CALL PCGSUB         ;green pattern set
0046                 ;
0047 F015 D1                POP  DE             ;D,E back (E = 15H)
0048 F016 14                INC  D              ;to next character
0049 F017 10F0              DJNZ LOOP           ;loop by counter B reg (4 char)
0050                 ;
0051                 ;
0052                 ;----- end of main routine -----
0053                 ;
0054 F019 C30010     EXIT:  JP   MONHOT         ;goto monitor
0055                 ;
0056                 ;
0057                 ;----- PCG pattern data -----
0058                 ;
0059                 ;****** data for code 30H ******
0060                 ;
0061 F01C 01020408 DATA:   DB   01H,02H,04H,08H,10H,20H,40H,80H
     F020 10204080
0062 F024 01020408         DB   01H,02H,04H,08H,10H,20H,40H,80H
     F028 10204080
0063 F02C 01020408         DB   01H,02H,04H,08H,10H,20H,40H,80H
     F030 10204080
0064                 ;
0065                 ;****** data for code 31H ******
0066                 ;
0067 F034 01020408         DB   01H,02H,04H,08H,10H,20H,40H,80H
     F038 10204080
0068 F03C 00000000         DB   00H,00H,00H,00H,00H,00H,00H,00H
     F040 00000000
0069 F044 00000000         DB   00H,00H,00H,00H,00H,00H,00H,00H
     F048 00000000
0070                 ;
0071                 ;****** data for code 32H ******
0072                 ;
0073 F04C 00000000         DB   00H,00H,00H,00H,00H,00H,00H,00H
     F050 00000000
0074 F054 01020408         DB   01H,02H,04H,08H,10H,20H,40H,80H
     F058 10204080
0075 F05C 00000000         DB   00H,00H,00H,00H,00H,00H,00H,00H
     F060 00000000
0076                 ;
0077                 ;****** data for code 33H ******
0078                 ;
0079 F064 00000000         DB   00H,00H,00H,00H,00H,00H,00H,00H
     F068 00000000
0080 F06C 00000000         DB   00H,00H,00H,00H,00H,00H,00H,00H
     F070 00000000
0081 F074 01020408         DB   01H,02H,04H,08H,10H,20H,40H,80H
     F078 10204080
0082                 ;
0083                 ;
0084 F07C                   END
```

これらのサンプル・プログラムにより、キャラクタ・ジェネレータについての理解を深めて下さい。

1-15 CG関係のシステム・サブルーチン

HuBASICでは、キャラクタ・ジェネレータ（CG）関係の命令として、DEFCHR＄とCGPAT＄があります。これらの命令処理で参照されるのが、次のシステム・サブルーチンたちです。

名　　称	PCGデータ・セット
アドレス	002BH(あるいは0AAAH)
入力条件	Dレジスタ　＝キャラクタ・コード Eレジスタ　＝CG I/Oアドレス上位8ビット(15H、16H、17Hのいずれか) HLレジスタ＝出力データ(8バイト)の格納先頭アドレス (HL)〜　　＝出力データ(8バイト分)
出力条件	HL←HL＋8 BCは保存
機　　能	PCG(B、R、G)の1つを選択して、(HL)〜(HL＋7)に格納されている8バイトデータをキャラクタ・コード(Dレジスタ)に出力する。

名　　称	CGデータ・リード
アドレス	0033H(あるいは0ACAH)
入力条件	Dレジスタ　＝キャラクタ・コード Eレジスタ　＝CG I/Oアドレス上位8ビット(14H、15H、16H、17H) HLレジスタ＝入力データ(8バイト)の格納先頭アドレス
出力条件	(HL)〜(HL＋7)に、CGから読んだデータを格納 HL←HL＋8 BCは保存
機　　能	CGの1つ(ROMCG、PCG(B、R、G))を選択して、Dレジスタの指定するキャラクタ・コードのパターンを読み、(HL)〜(HL＋7)のメモリーに格納する。

HuBASIC使用時には、これらのシステム・サブルーチンを利用するのが実用的でしょう。皆さんもDEFCHR＄やCGPAT＄に相当する処理をマシン語で実行してみて下さい。

第2章　グラフィック画面

2−1　グラフィック画面とGRAM

グラフィック画面への表示は、**グラフィックVRAM**（以下、GRAMと略称）にドット・パターンを書き込むことにより行なわれます。

X1C, X1Dでは、GRAMは標準実装されていて、X1シリーズの優れたグラフィック機能をすぐに楽しむことができます。一番最初のX1(CZ-800C)のみ、GRAMは別売(CZ-8GR)ですが、X1にとって「必需品」ともいえるデバイスですので、できるだけ取りつけることをお勧めします。

さて、GRAMは48Kバイトの容量を持ち、I／O空間内の4000H番地〜FFFFH番地に割りつけられています。BRGの3原色に応じて、GRAMは3つの部分に分けられ、I／Oマップは次のようになっています。

《GRAMのI／Oマップ》

アドレス	
4000H〜7FFFH	GRAM 1 (B)
8000H〜BFFFH	GRAM 2 (R)
C000H〜FFFFH	GRAM 3 (G)

グラフィック画面の構成単位は小さな点（**画素、ドット、ピクセル**ともいう）であって、40桁表示（WIDTH 40）の時、1画面は横320×縦200ドット、80桁表示（WIDTH 80）の時は、横640×縦200ドットから構成されます。

Hu BASICでは、グラフィック画面上のドットの位置を指定するのに0≦x≦319(又は639)、0≦y≦199の範囲の画面座標（x, y）を用います。これらの各位置とGRAMのアドレスとの対応は、テキスト画面より複雑です。

1ドットは、GRAMの1ビットに対応し、画面上左から8ドットずつ横1列が組になって、GRAMの1つの番地(1バイト)と対応します。たとえば、青画面(40桁表示)とGRAMアドレスとの対応を図示すると次のようになります。

青画面と GRAM アドレス（40 桁表示）

4000H	4001H			4026H	4027H
4800H	4801H			4826H	4827H
5000H	5001H			5026H	5027H
5800H	5801H			5826H	5827H
6000H	6001H			6026H	6027H
6800H	6801H			6826H	6827H
7000H	7001H			7026H	7027H
7800H	7801H			7826H	7827H
4028H	4029H			404EH	404FH
4828H	4829H			484EH	484FH
5028H	5029H			504EH	504FH
5828H	5829H			584EH	584FH
6028H	6029H			604EH	604FH
6828H	6829H			684EH	684FH
7028H	7029H			704EH	704FH
7828H	7829H			784EH	784FH
43C0H	43C1H			43E6H	43E7H
4BC0H	4BC1H			4BE6H	4BE7H
53C0H	53C1H			53E6H	53E7H
5BC0H	5BC1H			5BE6H	5BE7H
63C0H	63C1H			63E6H	63E7H
6BC0H	6BC1H			6BE6H	6BE7H
73C0H	73C1H			73E6H	73E7H
7BC0H	7BC1H			7BE6H	7BE7H

画面座標（x，y）から、GRAM アドレスへの換算を BASIC プログラムにしてみました。以下のプログラムで、140 行～160 行に「初期値」を設定してから RUN すると、画面の各ドットに対応する GRAM アドレスが計算できます。40 桁表示の時は、ページ 0 とページ 1 がありますので注意して下さい。

```
100 REM --- GRAM address ---
110 '
120 DEFINT A-Z
130 '
140 START=&H4000   :REM <— GRAM (B)
150 WMODE=40       :REM <— WIDTH value
160 PAGE =0        :REM <— page 0
170 '
180 INPUT "(X,Y)=";X,Y
190 XX=(X ¥ 8) : YY=(Y ¥ 8) : DAN=(Y MOD 8)
200 GRAM=START+PAGE*&H400+YY*WMODE+XX+DAN*&H800
210 PRINT "GRAM = ";HEX$(GRAM)+"H"
220 PRINT
230 GOTO 180
```

```
RUN
(X,Y)=? 7,0
GRAM = 4000H
(X,Y)=? 8,0
GRAM = 4001H
(X,Y)=? 0,192
GRAM = 43C0H
(X,Y)=? 319,199
GRAM = 7BE7H
(X,Y)=? ■
```

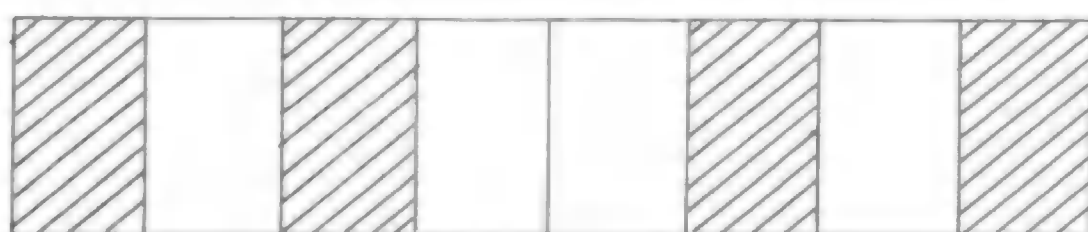
こうして、基本3色(青、赤、緑)の画面とGRAMの対応はわかりました。これ以外の色は、基本色の重ね合わせで実現できますから、たとえば青画面と赤画面の同一位置にドットをセットすると、それは赤紫(マゼンタ)のドットを表示したことになる訳です。

［グラフィック画面の実験の注意］

LISTやAUTOを実行すると、グラフィック画面が見えなくなることがあります。これは「グラフィック表示スイッチ」(第7節参照)がOFFとなっているからです。このことに留意し、以下の各節でグラフィック画面への表示実験をする時には、最初INITあるいはPALETあるいはSCREEN 0, 0, 0を実行して「グラフィック表示スイッチ」をONにしてから始めることをお勧めします。プログラムを実行したのに表示されない時は、まず、このことを疑ってみて下さい。

2-2 GRAMへのデータ出力

次は、GRAMへの出力データを考えます。テキストVRAMのときは、出力データはキャラクタのASCIIコードでしたが、GRAMの場合はどうでしょうか。

2進数のビット・パターンで考えて、「0＝ドットリセット, 1＝ドットセット」というのが答です。たとえば、GRAMの4000H番地にデータA5Hを出力すると、画面左上隅に点線のパターンが青く表示されます。A5Hは2進表示で10100101ですから画面には、

(斜線部はドット・セット)

というドット・パターンとなって表示された訳ですね。表示位置は前節のことから、画面座標（0，0）〜（7，0）の8ドットとなりますね。

では、基本テクニックとして、40桁表示・ページ0（WIDTH 40：SCREEN 0，0，0）のグラフィック画面において、Hu BASICでの

```
LINE (0, 1)- (319, 1), PSET, 1
```

にあたるプログラムを作りましょう。ループを用いて次のようになるはずです。

マシンコード	ニーモニック
01 00 48	LD BC, 4800H ;位置(0, 1)
3E FF	LD A, 0FFH ;実線
16 28	LD D, 40 ;1行=40バイト
ED 79	LOOP:OUT (C), A ;PSET
03	INC BC ;アドレス+1
15	DEC D ;カウンター1
20 FA	JR NZ, LOOP ;
C9	RET

「リロケータブル」なので、適当なアドレスから格納して実行しましょう。いかがですか？　青い線がひけましたか？

《青い線をひく》

```
MON
*ME000
:E000=010048
:E003=3EFF
:E005=1628
:E007=ED79
:E009=03
:E00A=15
:E00B=20FA
:E00D=C9
:E00E=00
*GE000
*■
```

グラフィック画面表示といっても、このようにGRAMに1つ1つデータを送っていくことが基礎になる訳ですね。

2-3　テキスト画面からグラフィック画面への転送

では、もう少しレベルを上げて、テキスト画面の1文字のフォント・パターン（8×8ドット）をグラフィック画面に表示することを考えましょう。

サンプルプログラムでは、アットマーク（@）のフォント・パターンをROMCGから読み出し、赤画面の左上隅に表示しています。GRAM(R)に描かれている証拠に CLR を押しても消えませんね。

```
100 REM ************************
110 REM *                      *
120 REM *  text —> graphic     *
130 REM *                      *
140 REM *    by Y.Shimizu      *
150 REM *                      *
160 REM *     1984.7.16        *
170 REM *                      *
180 REM ************************
190 '
200 CLEAR &HE000
210 WIDTH 40 : SCREEN 0,0,0 : CLS 4
220 '
230 GOSUB "ｶｷｺﾐ"
240 CALL &HE000
250 END
260 '
270 '
280 LABEL "ｶｷｺﾐ" : REM ----------------------------------------------
290 '
300 ADR=&HE000
310 READ MC$
320 IF MC$="END" THEN RETURN
330 POKE ADR,VAL("&H"+MC$)
340 ADR=ADR+1
350 GOTO 310
360 '
370 REM --- ﾏｼﾝｺﾞ ﾙｰﾁﾝ -----------------------------------------------
380 '
390                   :REM CGRD:  EQU  0033H
400                   :REM ;
410                   :REM        ORG  0E000H
420                   :REM ;
430 DATA 16,40       :REM CGPAT: LD   D,40H      ;code '@'
440 DATA 1E,14       :REM        LD   E,14H      ;ROMCG
450 DATA 21,00,E1    :REM        LD   HL,0E100H  ;data
460 DATA CD,33,00    :REM        CALL CGRD
470                   :REM ;
480 DATA 11,00,E1    :REM XFER:  LD   DE,0E100H  ;data
490 DATA 01,00,80    :REM        LD   BC,8000H   ;GRAM(R)
500 DATA 21,00,08    :REM        LD   HL,0800H   ;step
510 DATA 3E,08       :REM        LD   A,8        ;counter
520 DATA F5          :REM LOOP:  PUSH AF
530 DATA 1A          :REM        LD   A,(DE)     ;A=data
540 DATA ED,79       :REM        OUT  (C),A      ;out to GRAM
550 DATA 13          :REM        INC  DE         ;data pointer inc
560 DATA E5          :REM        PUSH HL
570 DATA 09          :REM        ADD  HL,BC
580 DATA 44          :REM        LD   B,H        ;BC=BC+800H
590 DATA 4D          :REM        LD   C,L
600 DATA E1          :REM        POP  HL
610 DATA F1          :REM        POP  AF         ;A=counter
620 DATA 3D          :REM        DEC  A          ;counter dec
630 DATA 20,F2       :REM        JR   NZ,LOOP
640 DATA C9          :REM        RET
650                   :REM ;
660 DATA END,        :REM end mark
670 '
680 REM -------------------------------------------------------------
```

2－4 「低速」画面クリア

次は、グラフィック3画面（B，R，G）をすべて消去するプログラムを作りましょう。単純に考えると、GRAM全部(4000 H～FFFFH)にデータ00 Hを出力すればよさそうです。

《低速画面クリア》

マシンコード	ニーモニック
01 00 40	LD BC, 4000H ;GRAM先頭アドレス
AF	LOOP: XOR A ;A＝00H
ED 79	OUT (C), A ;1バイト・クリア
03	INC BC ;アドレス＋1
78	LD A, B ;BC＝0000？
B1	OR C
20 F8	JR NZ, LOOP
C9	RET

```
MON
*MF000
:F000=010040
:F003=AF
:F004=ED79
:F006=03
:F007=78
:F008=B1
:F009=20F8
:F00B=C9
:F00C=00
*R
Ok
LINE(0,0)-(319,199),PSET,7,BF:CALL&HF000
```

上では、メモリー上　F 000 H～F 00 BH番地にプログラムを格納しています。画面は40桁表示にしておきました。画面消去の様子を観察するために、画面を白く塗りつぶしておいてから、マシン語プログラムを実行します。

結果は、「ホーワッ」とした感じで画面消去がなされるはずです。途中で、白が黄色、緑色に変色していくのを目で追うことができて、これでは決して高速とは言えません。Hu BASICのCLS 0では、もっと機敏に「サッ」と消えたはずです。上のプログラムはそんなに「無駄」をしてないはずですが……。どこが違うのでしょうか？

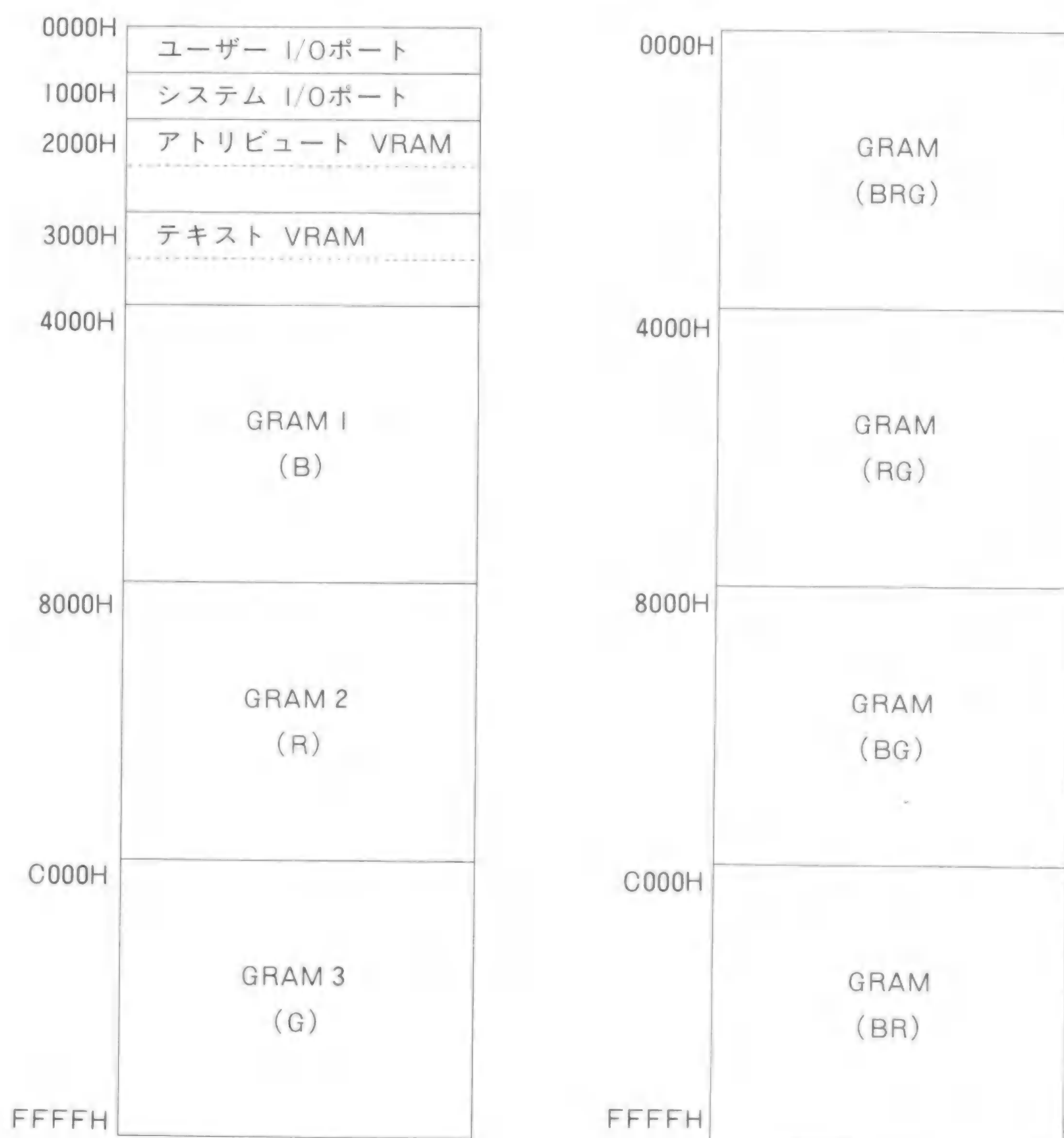
悩むのも当然です。原因は、ソフトウェアというよりも、ハードウェアの方にあるからです。この問題を論ずるには、GRAMアクセスモード切替えのテクニックを知る必要があります。次節ではこれを扱いましょう。

2－5　アクセス・モードの切り替え

前節までは簡単のため、わざと避けて来たことなのですが、実は、X１シリーズにおいては、GRAMのI／Oマップは２種類存在します。今まで登場してきたI／Oマップは、B，R，Gの各GRAMを個別にアクセスする場合で、**個別アクセス・モード**あるいは**シングル・アクセス・モード**とよばれます。これに対して、複数のGRAMを同時にアクセスできるモードがあり、**同時アクセス・モード**とよばれます。これら２つのアクセス・モードで、I／Oマップは次のように異なります。

２つのアクセス・モードとI／Oマップ

アドレス	個別アクセス・モード
0000H	ユーザー I/Oポート
1000H	システム I/Oポート
2000H	アトリビュート VRAM
3000H	テキスト VRAM
4000H	GRAM 1 (B)
8000H	GRAM 2 (R)
C000H	GRAM 3 (G)
FFFFH	

アドレス	同時アクセス・モード
0000H	GRAM (BRG)
4000H	GRAM (RG)
8000H	GRAM (BG)
C000H	GRAM (BR)
FFFFH	

同時アクセス・モードにおいては、何と、I／O空間がすべてGRAMで埋まってしまいます。このモードではもちろんテキストVRAMへのアクセスはできないし、1000H～1FFFH番地にあるシステム用のI／Oポートも使えません。Hu BASICのコマンドで、グラフィック３画面をクリアする　CLS 0　が高速消去するのは、同時アクセス・モードでGRAM（BRG）をクリアしているからなのです。

では、これら2つのアクセス・モード間の切り替えは、どのようにして行なうのでしょうか。まず、個別アクセス・モードにするのは簡単で、ダミーの入力命令　IN A,(C)を1回実行すればよいのです。

面倒なのは、同時アクセス・モードへの切り替え法です。I／Oポートの　1A02H番地のビット5にパルスを出力すると同時アクセス・モードになります。具体的には次のようにします。

《同時アクセス・モードへの切り替え法》

```
LD      BC, 1A02H   ;I/Oアドレスの設定。
IN      A,(C)       ;1A02H番地を読む。
SET     5, A        ;ビット5を1にする。("H")
OUT     (C), A      ;1A02H番地へ出力。
RES     5, A        ;ビット5を0にする。("L")
OUT     (C), A      ;1A02H番地へ出力。
```

これを利用すると、CLS 0に相当する高速画面クリアは、次のようにプログラムできます。

高速画面クリア

マシンコード	ニーモニック
F3	DI ;割り込み禁止
01 02 1A	MULACS: LD BC, 1A02H ;同時アクセス・モードへ切り替え
ED 78	IN A, (C)
CB EF	SET 5, A
ED 79	OUT (C), A
CB AF	RES 5, A
ED 79	OUT (C), A
01 00 40	CLS0: LD BC, 4000H ;GRAM(BRG)終了アドレス+1
AF	LOOP: XOR A
0B	DEC BC ;アドレス−1
ED 79	OUT (C), A ;1バイト・クリア
78	LD A, B ;BC=0000H?
B1	OR C
20 F8	JR NZ, LOOP
0B	SNGACS: DEC BC ;BC=FFFFH
ED 78	IN A, (C) ;個別アクセス・モードへ切り替え
FB	EI ;割り込み許可
C9	RET

同時アクセス・モードで　0000H～3FFFH番地に00Hを出力するものですが、X1ではキー入力などを割り込みで処理しているので、プログラム実行中に割り込みがかかると、システムは普通のI／Oマップと見なして、I／Oポートをアクセスしますから変なことになってしまいます。そこで、最初にDIを実行し、プログラム動作中の割り込みを禁止します。

プログラムの最後の方で、ダミーのIN命令を実行し、個別アクセス・モードに戻します。ループ終了時は BC=0000 H となっていますが、ここは拡張I／Oポートに割り当てられたアドレスで、何が接続されるかわかりませんから、安全のため DEC BC により、BC=FFFFH にして IN A,(C) を実行しています。

プログラム終了前に EI で割り込み許可に戻しておくのも忘れずに！

実行例を下に掲げます。

《画面クリアの実行例》

```
MON
*MF000
:F000=F3
:F001=01021A
:F004=ED78
:F006=CBEF
:F008=ED79
:F00A=CBAF
:F00C=ED79
:F00E=010040
:F011=AF
:F012=0B
:F013=ED79
:F015=78
:F016=B1
:F017=20F8
:F019=0B
:F01A=ED78
:F01C=FB
:F01D=C9
:F01E=00
*R
Ok
LINE(0,0)-(319,199),PSET,7,BF:CA.&HF000
```

前節と同様F000H番地から配置してみました(「リロケータブル」なので別の番地に配置することも可能)。前節の「低速画面クリア」との速度の違いを味わって下さい。

第4章で「汎用ポート8255」の使い方を理解すると、同時アクセス・モードへの切り替えを別の形でプログラムできることがわかります。その時、本節の内容を思い出して下さい。

2－6　パレット設定

X1シリーズのグラフィック画面は「パレット機能」を持っていて、画面に表示される時の色を、本来の色と違った色に設定できます。これは画面出力の出口の所で、切り換えスイッチがあり、本来の色とは異なる色の出力回路に色コードを入れられるからです。HuBASICの命令 PALET, CANVAS はこの機能をサポートした命令です(マニュアルに、この2つの命令は同時に使用しないよう注意が書かれているのはそのためです)。

パレット機能には、次の3つのI／Oアドレスが関係しています。

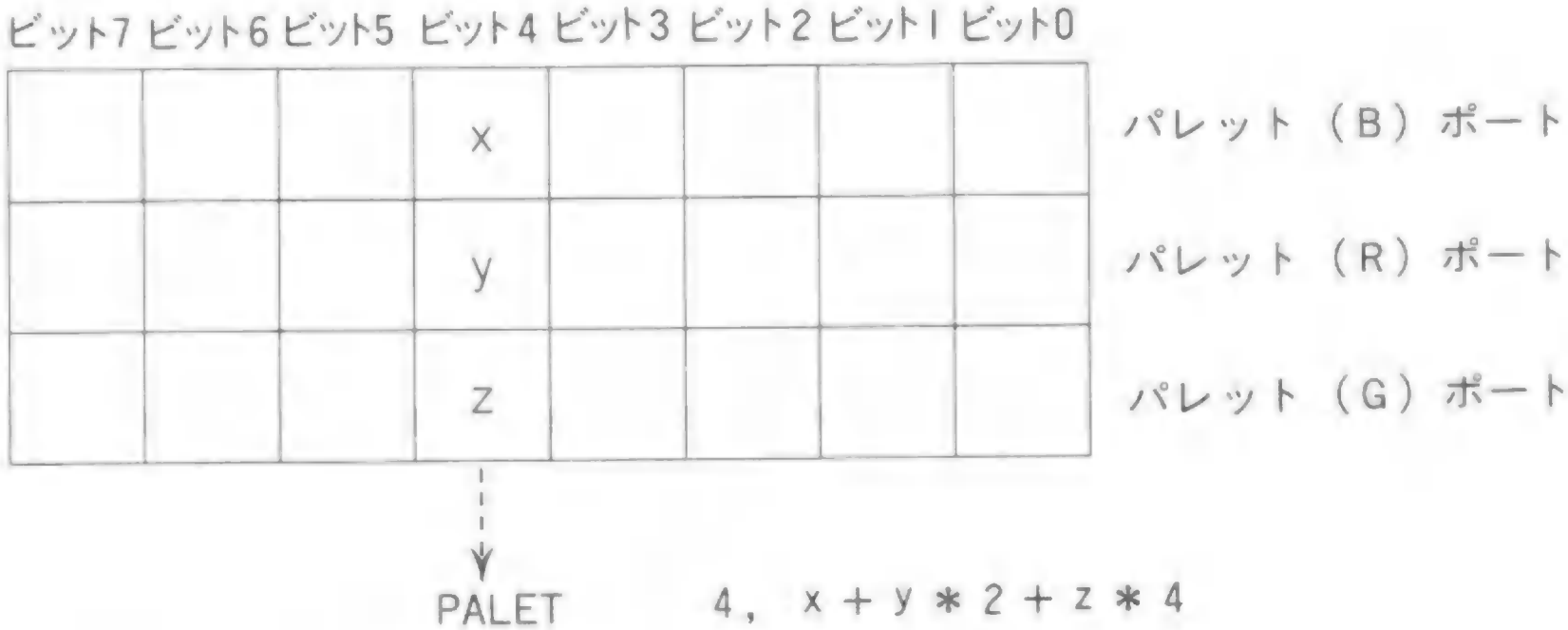
パレット関係I／Oポート

I/Oアドレス	機　　能
1000H	パレット(Blue)　設定ポート
1100H	パレット(Red)　設定ポート
1200H	パレット(Green) 設定ポート

〔注〕これらのI／Oアドレスは、アドレス上位バイトのみ有効なので、下位バイトは何でもよい。たとえば、1203H番地は1200H番地と同じ機能をもつ。

これら、3つのポートは組になって働き、書き込まれるデータと設定されるパレットの関係は次のようになります。

《パレットの設定》

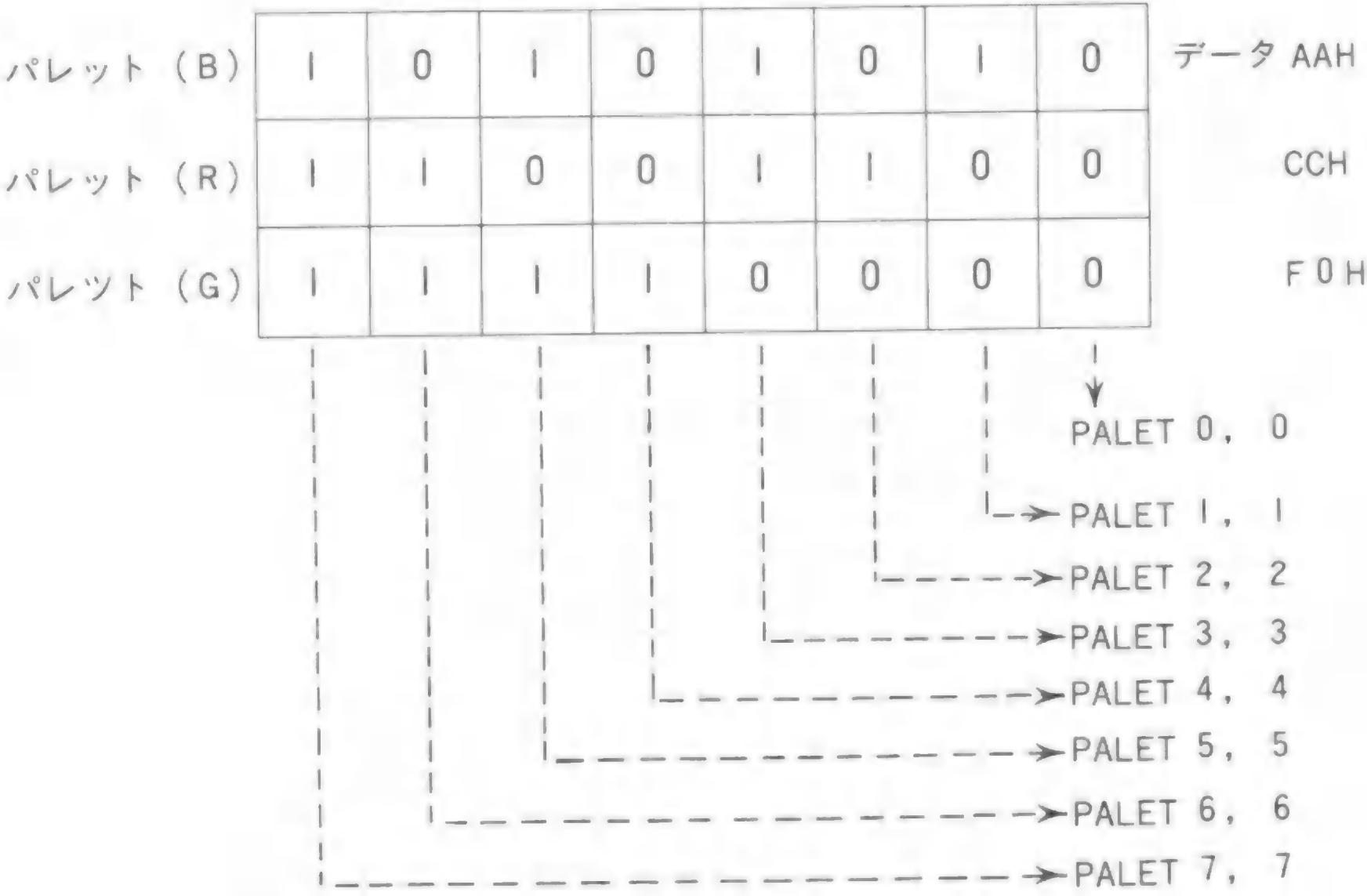
ビット7	ビット6	ビット5	ビット4	ビット3	ビット2	ビット1	ビット0	
			x					パレット（B）ポート
			y					パレット（R）ポート
			z					パレット（G）ポート

PALET　4，x＋y＊2＋z＊4

3つのパレット・ポートを並べて、ビットごとに縦割りに見ます。図のように、たとえばビット4の列に上からx，y，z（各々は0か1）があるとすると、HuBASICでいえばPALET 4，x＋y＊2＋z＊4 の状態と同じになって、色コード4番には（x＋y＊2＋z＊4）番の表示色が設定されます。

パレットの初期設定は次のようになっています。

《パレット初期設定》

パレット（B）	1	0	1	0	1	0	1	0	データ AAH
パレット（R）	1	1	0	0	1	1	0	0	CCH
パレツト（G）	1	1	1	1	0	0	0	0	F0H

PALET 0，0
PALET 1，1
PALET 2，2
PALET 3，3
PALET 4，4
PALET 5，5
PALET 6，6
PALET 7，7

これらのデータを変更すれば、パレット設定が変わります。実際は、コンピュータ画面出力の垂直同期信号（PV-Sync）とタイミングをとって、データ出力をします。たとえば、PALET 4，1 を実現するには、次のようにデータをセットします。

《PALET 4， 1》

パレット (B)	1	0	1	1	1	0	1	0	データ BAH
パレット (R)	1	1	0	0	1	1	0	0	CCH
パレット (G)	1	1	1	0	0	0	0	0	E0H

PALET 4， 1 （他のパレットは変更なし）

プログラムは次のようになります。

《PALET 4， 1プログラム》

マシン・コード	ニーモニック
01 01 1A	VSYNC:LD BC, 1A01H ;垂直同期待ち
ED 78	LOOP: IN A, (C)
CB 57	BIT 2, A ;ビット2=PV-Sync
28 FA	JR Z, LOOP
01 00 10	PALET:LD BC, 1000H ;BC=1000H
3E BA	LD A, 0BAH ;パレット(B)データ
ED 79	OUT (C), A
04	INC B ;BC=1100H
3E CC	LD A, 0CCH ;パレット(R)データ
ED 79	OUT (C), A
04	INC B ;BC=1200H
3E E0	LD A, 0E0H ;パレット(G)データ
ED 79	OUT (C), A
C9	RET

垂直同期待ちは、I／Oポート1A01H番地のビット2を調べ、ビットが"H"なるのを待ちます。この後に3つのパレット・ポートへデータを出力します。効果を見るために40桁表示画面を緑色に塗りつぶしておきます。実行すると、一瞬に青色に変わり、 PALET 4， 1 となったことがわかるはずです。

《実行例》

```
PALET:LINE(0,0)-(319,199),PSET,4,BF
Ok
MON
*ME000
:E000=01011A
:E003=ED78
:E005=CB57
:E007=28FA
:E009=010010
:E00C=3EBA
:E00E=ED79
:E010=04
:E011=3ECC
:E013=ED79
:E015=04
:E016=3EE0
:E018=ED79
:E01A=C9
:E01B=00
*GE000
*■
```

実用的には、HuBASIC内のシステム・サブルーチンを利用するのがよいでしょう。

名　称	パレット設定ルーチン
アドレス	テープBASICのとき　3AD4H ディスクBASICのとき　3B06H
入力条件	(00F6H)=パレット(B)データ (00F7H)=パレット(R)データ (00F8H)=パレット(G)データ
機　能	ワークエリア(00F6H～00F8H番地)にセットされたデータに従って、パレットを設定する。

このルーチンを用いて次のようにしても、PALET 4, 1 が実行されます(ディスクの方は ＊G 3B06)。

《システム・サブルーチンの利用》

```
MON
*M00F6
:00F6=BA
:00F7=CC
:00F8=E0
:00F9=00
*G3AD4
*■
```

2-7 グラフィック表示スイッチ

本章第1節でも注意したように、LIST 命令や AUTO 命令では、グラフィック画面が見えなくなる現象が起こります。この原因は、LIST や AUTO では SCREEN というコマンドを処理するルーチンが実行されるからです。マニュアルに説明されているように、SCREEN のすべてのパラメータを省略すると、グラフィック画面が見えなくなります。処理ルーチンは、テープ BASIC の場合 3BD8H～3BEFH 番地、ディスク BASIC の場合は 3C0AH～3C21H 番地です。秘密を探るためこの部分を逆アセンブルすると、次のようになっています(アドレスはテープ BASIC に準拠しました)。

SCREEN 処理ルーチン
逆アセンブル リスト(テープ BASIC)

```
3BD8 11F800     LD   DE,00F8H
3BDB CD383B     CALL 3B38H
3BDE 010312     LD   BC,1203H
3BE1 1A         LD   A,(DE)
3BE2 E601       AND  01H
3BE4 2802       JR   Z,3BE8H
3BE6 3EFF       LD   A,0FFH
3BE8 ED79       OUT  (C),A
3BEA 1B         DEC  DE
3BEB 05         DEC  B
3BEC 0D         DEC  C
3BED 20F2       JR   NZ,3BE1H
3BEF C9         RET
```

すでに、私たちはパレットの設定を理解していますから、上記サブルーチンの意味を解析することができるはずです。すなわち、パラメータを省略した時の SCREEN の処理は、「すべてのパレット設定を第0パレットの色コードと同一にする」ことと解読できます。通常、第0パレットには色コード0（透明）が入っていますから、

```
FOR A=0 TO 7：PALET A, 0：NEXT
```

が実行されたのと同じになり、グラフィック画面が消えるのです。本節の標題は「グラフィック表示スイッチ」としてありますが、見かけがそう見えるだけであって、このようにプログラム上で処理している訳ですね。「幽霊の正体見たり枯尾花」といった所でしょうか。

従って、原理を逆用すると、面白い現象が起こります。次のように、第0パレットの色コードを1（青色）にしておくと、SCREEN 命令で全画面が青くなります。

《SCREEN でまっ青！》

```
MON
*M00F6
:00F6=AB
:00F7=CC
:00F8=F0
:00F9=00
*R
Ok
SCREEN
Ok
■
```

私はよく、X1のスーパー・インポーズを利用し、画面の下方にグラフィックで青い表示エリアを作り、この中でプログラムしながら、テレビを見たりするのですが、こんな時、LIST や AUTO で背景のグラフィック画面が消えては困ります。

これに対する解決策は、もうおわかりですね。一番簡単な方法は上記の SCREEN 処理ルーチンが実行されないように変えてしまうことです。次のようなわずか1バイトの変更で済みます。

「グラフィック表示スイッチ」をOFFしない法	
テープBASICのとき	3BD8H番地の内容を 11H→C9Hに変更
ディスクBASICのとき	3C0AH番地の内容を 11H→C9Hに変更

こうすれば、グラフィック画面を消さずに LIST や AUTO が実行できます。（元に戻す時は上記番地に 11H を書き込んで下さい。）

2-8 プライオリティ機能

HuBASICでは、PRWコマンドによりテキスト画面とグラフィック画面の優先順位を各色ごとに決めることができます。ここでは、**テキスト・プライオリティ**とよぶことにします。

テキスト・プライオリティは1バイトのデータにより設定され、その意味は次のようになっています。

ビット7	6	5	4	3	2	1	ビット0
白	黄	シアン	緑	マゼンタ	赤	青	黒

各ビットとも　0のときテキスト画面優先
　　　　　　　1のときグラフィック画面優先

このデータをI／Oポートの1300H番地に出力すると、テキスト・プライオリティが設定されます。

《関連I／Oポート》

I/Oアドレス	機　　　能
1300H	テキスト・プライオリティの設定ポート

パレット設定と同じく、垂直同期待ちをしてから、データを出力します。サンプル画面として以下を実行して下さい。

《サンプル画面》

```
CLEAR &HE000
Ok
10 FOR I=0 TO 7
20 LINE (40*I,0)-(40*I+39,199),PSET,I,BF
30 NEXT
40 END
RUN
Ok
■
```

さて、いよいよ次のマシン語プログラムを実行して、テキスト・プライオリティ機能を確認しましょう。

```
MON
*ME000
:E000=01011A
:E003=ED78
:E005=CB57
:E007=28FA
:E009=010013
:E00C=3EAA
:E00E=ED79
:E010=C9
:E011=00
*R
Ok
FOR I=0 TO 39:PRINT"A";:NEXT:CALL&HE000
AAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAA
Ok
```

テキスト画面の例として、文字"A"を横一列に表示してから、マシン語ルーチンをコールすると、1色おきに"A"がグラフィックの裏に隠れます。

これも、実用上はシステム・サブルーチンを利用するのが便利です。

名　　称	テキスト・プライオリティ設定ルーチン
アドレス	テープBASICのとき　　3B2BH ディスクBASICのとき　3B5DH
入力条件	Eレジスタ＝テキスト・プライオリティ・データ
出力条件	ワークエリア(00F9H)←プライオリティ・データ
機　　能	Eレジスタのデータに従って、テキスト・プライオリティを設定する。その際、BASICのワーク・エリア00F9H番地を書き替える。

第3章　サウンド機能

3－1　サウンド機能とPSG

X 1 シリーズには、AY-3-8910 と呼ばれるサウンド用専用LSI（Programmable Sound Generator 略してPSGという）が搭載されていて、3 和音 1 雑音を出すことができます。

HuBASICでは、音出し関係の命令として、MUSIC、PLAY、SOUND、BEEP がありますが、これらはすべて、上記PSGを操作することにより音を出しています。本章の内容は、BASICコマンドのSOUND文をキッチリ理解することが目的です。実際この命令は、直接PSGへデータを送るものなので、ほとんどマシン語的感覚のコマンドといえます。

3－2　PSGのレジスタ

PSGは、0番から15番まで16個のレジスタを持つLSIで、このうち0番～13番が音出し関係、14～15番が入出力ポートにあてられています（X 1ではジョイスティック用ポートとして用いています）。各レジスタの概容と内部構成は次の通りです。

PSG（AY-3-8910）のレジスタ

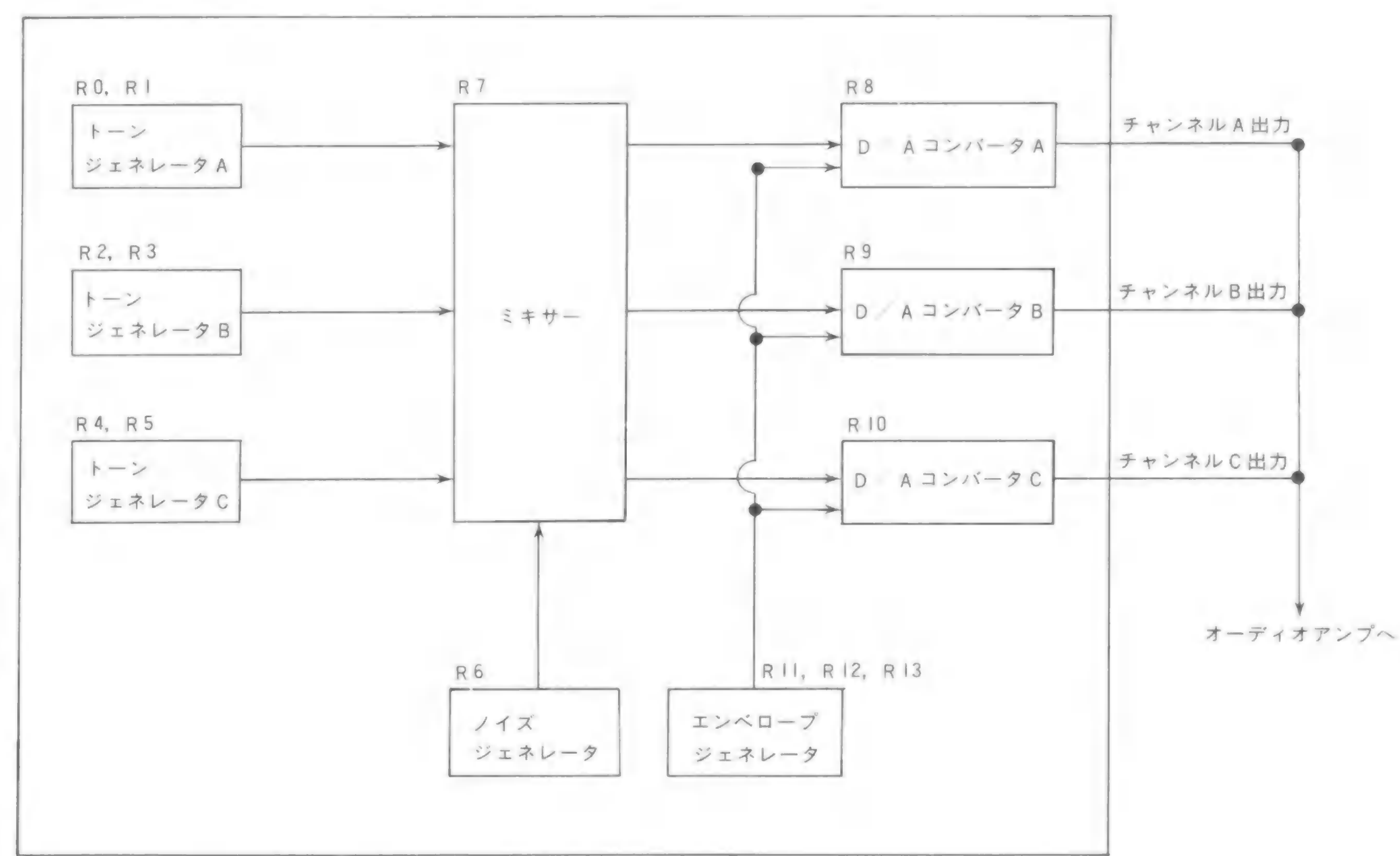
レジスタ番号	機　　能	データの範囲
0	チャンネルA周波数(トーン) 微調整	0～255
1	〃 粗調整	0～15
2	チャンネルB周波数(トーン) 微調整	0～255
3	〃 粗調整	0～15
4	チャンネルC周波数(トーン) 微調整	0～255
5	〃 粗調整	0～15
6	ノイズ周波数	0～31
7	ミキサー(トーン、ノイズON/OFF)	0～63〔注1〕

レジスタ番号	機　　能		データの範囲
8	チャンネルAの音量		0～15〔注2〕
9	チャンネルB 〃		0～15〔注2〕
10	チャンネルC 〃		0～15〔注2〕
11	エンベロープ周期	微調整	0～255
12	〃	粗調整	0～255
13	エンベロープ・パターン		0～15
14	入出力ポートA		〔注3〕
15	入出力ポートB		〔注3〕

［注1］レジスタ 14、15 のモード指定機能もあり、その場合には、この範囲外のデータが有効となる。

［注2］データ 16 も意味があり、その場合には、エンベロープ指定となる。

［注3］X1では、ジョイスティック・ポートおよびデジタル・テロッパー用ポートとして使用する。

PSGの内部構成

AY-3-8910

［注］Rnは「レジスタn」のこと。

3－3　トーン・ジェネレータ

PSGには3基の**トーン・ジェネレータ**（音発生器）が組み込まれています。R 0～R 5が、これらトーン・ジェネレータを制御するレジスタです（1基につき、2つのレジスタを持つ)。これらのレジスタにデータを書き込むと、音程を決めることができます。

PSGは、4 MHzのCPUクロックを分周した、2 MHzのクロックを基本周波数 fo として持っています。これは内部でさらに 1/16 にされ、この周波数 fo/16 が、トーン・ジェネレータにより分周されて、目的の音程を作る訳です。R 0、R 2、R 4に設定する値は微調整データですから、FT で表わします(Fine Tuningの略)。また、R 1、R 3、R 5の値は粗調整データですから、CT で表わします（Coarse Tuningの略)。このとき、トーン・ジェネレータから出力される周波数 f は次式で求まります。

$$f = \frac{fo}{16 \times (CT \times 256 + FT)} \quad , \; fo = 2000000(2MHz)$$

逆に、出力したい周波数がわかっている時、CT, FT の値を求めるには、BASICの関数を用いて表わすと、

```
CT=INT ((fo ／(16 * f ))／ 256)
FT=FRAC ((fo ／(16 * f ))／ 256) * 256
```

とすればよいのです（ CT は0～15の値が有効です)。

HuBASICでいうと MUSIC"O 4 A" により出る音は、オーケストラが音合せなどに用いる音だそうですが、この音は440 Hzになっています。トーン・ジェネレータから440 Hzの周波数の音を出すための FT, CT の値を計算すると、

```
? 2000000/(16*440)
 284.09091
Ok
CT=INT(284/256)
Ok
FT=FRAC(284/256)*256
Ok
?CT
 1
Ok
?FT
 28
Ok
■
```

となります（ fo ／(16 * f) の部分を先に計算しておきました)。

こうして、たとえばチャンネルAのトーン・ジェネレータの周波数を440 Hzに設定したければ、SOUND文で次を実行すればよいことになります。

```
SOUND 0、28：SOUND 1、1
```

参考までに、"O 4 C"～"O 6 G"までの音を出したい時に、FT, CT に設定する値を書いておきます。ゲームなどに使うと便利だと思います。

おもな音のFTとCT

音記号	FT	CT	音記号	FT	CT
O4C	221	1	O5E	189	0
O4#C	195	1	O5F	176	0
O4D	169	1	O5#F	168	0
O4#D	145	1	O5G	159	0
O4E	123	1	O5#G	150	0
O4F	101	1	O5A	142	0
O4#F	81	1	O5#A	134	0
O4G	62	1	O5B	126	0
O4#G	45	1	O6C	119	0
O4A	28	1	O6#C	112	0
O4#A	12	1	O6D	106	0
O4B	253	0	O6#D	100	0
O5C	238	0	O6E	94	0
O5#C	225	0	O6F	89	0
O5D	212	0	O6#F	84	0
O5#D	200	0	O6G	79	0

3－4　ノイズ・ジェネレータ

PSGは音源として、3基のトーン・ジェネレータ以外に、**ノイズ・ジェネレータ**（雑音発生器）を1基持っています。R6がこのためのレジスタです。

ノイズ・ジェネレータの発生する雑音の周波数 f_N と、R6に設定される値（Noise Tuningの頭文字をとってNTとする）の関係式は次の通りです。

$$f_N=\frac{fo}{16\times NT}\quad ,\ fo=2000000(2MHz)$$

ただし、NT＝0 のときは、$f_N=fo/16$ と約束しておきます。NTは5ビット数値で 0～31の範囲が有効です。NTの値を小さくすると、周波数 f_N が高くなり、「シャー」という音になります。逆にNTを大きくすると、f_N が低くなり、「ザァー」と聴えます。

3－5　ミキサー

PSGには、A，B，Cの3チャンネルがあると言いましたが、各チャンネルから、トーンを出すか、ノイズを出すかを独立に決めるためのスイッチの働きをしているのがミキサー部分で、レジスタR7により制御できます。R7の各ビットの意味は次の通りです。

《R 7のビットの意味》

ビット7	ビット6	ビット5	ビット4	ビット3	ビット2	ビット1	ビット0
〔注〕	〔注〕	チャンネル C ノイズ	チャンネル B ノイズ	チャンネル A ノイズ	チャンネル C トーン	チャンネル B トーン	チャンネル A トーン

各ビットは0のときON（音を出す）
1のときOFF（音を出さない）

［注］　R 14、R 15の入出力ポートの指定に関するもので、第10節を見て下さい。

X1には&Bという2進数指定機能があるので、これを使うと便利です。

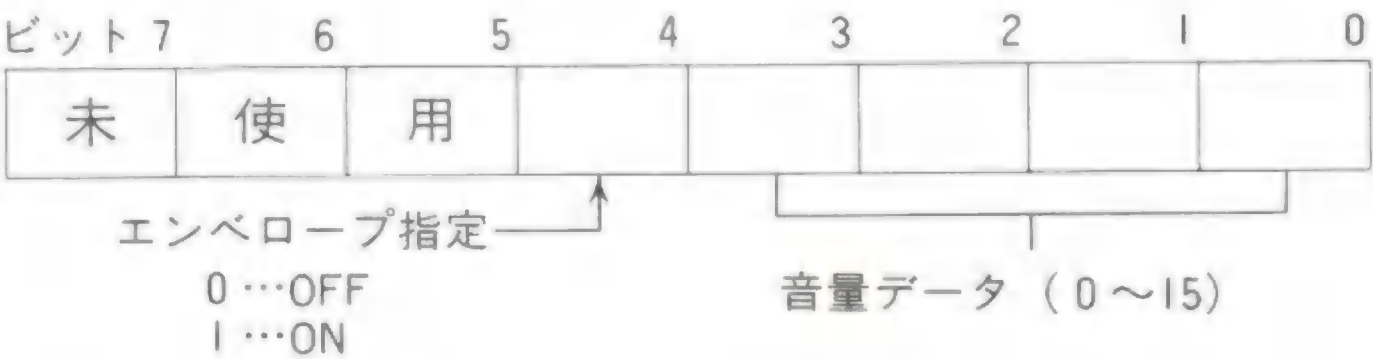
［例］

《2進表示》	《10進表示》	《意味》
SOUND 7，&B111111	SOUND 7，63	3チャンネルとも音は出さない(ミュート)。
SOUND 7，&B000111	SOUND 7，7	3チャンネルともノイズにする。
SOUND 7，&B111000	SOUND 7，56	3チャンネルともトーンにする。
SOUND 7，&B110001	SOUND 7，49	Aのみノイズ，B，Cはトーン。
SOUND 7，&B000000	SOUND 7，0	3チャンネルで，ノイズ・トーンが合成される。

3－6　D／Aコンバータ

D／Aコンバータは「音の出口」にあって、トーン・ジェネレータとノイズ・ジェネレータから送り出されてくる「デジタル・オーディオ信号」を、スピーカを動かすための「アナログ・オーディオ信号」に変換する働きと、音量の制御（アンプ）機能を受け持つ部分です。各チャンネルに対応して、計3基あって、R 8～R 10がそのレジスタです。

R 8～R 10の各レジスタとも5ビットが有効で、その内容は次の通りです。

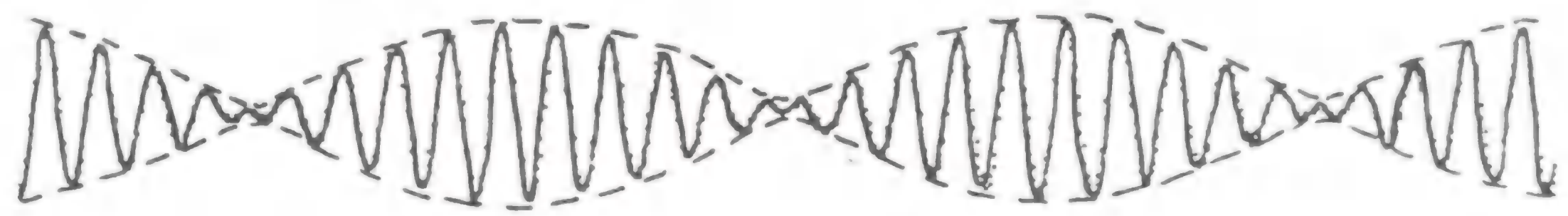
ビット7	6	5	4	3	2	1	0
未	使	用	エンベロープ指定 0…OFF 1…ON	音量データ（0～15）			

ただし、エンベロープONのとき、音量データは無効となる。通常は0にしておく。

音量は0のときがミュート、15のときが最大です。ただし、データとして16を与えると、後述するエンベロープ・パターンに従って音量が変化します。

各チャンネル独立に、音量指定ができますから、主旋律にあわせて副旋律を小さく奏でる等のことが可能です。

3－7 エンベロープ・ジェネレータ

英和辞典で envelope を引くと、「封筒、包み」と出ています。数学では、さらに意味が派生して「包絡線」の意味で使われます。

《包絡線（エンベロープ)》

上のような小刻みに振動するグラフがあるとき、その輪郭を「包む」点線を包絡線とよびます。音で使われるエンベロープも、どうもこの意味から来ているようで、「音量の時間的変化」のことです。

PSG では、エンベロープのパターンとして、次の8種類を使うことができます。これらはR 13によって選択され、0～15のコードと対応します。

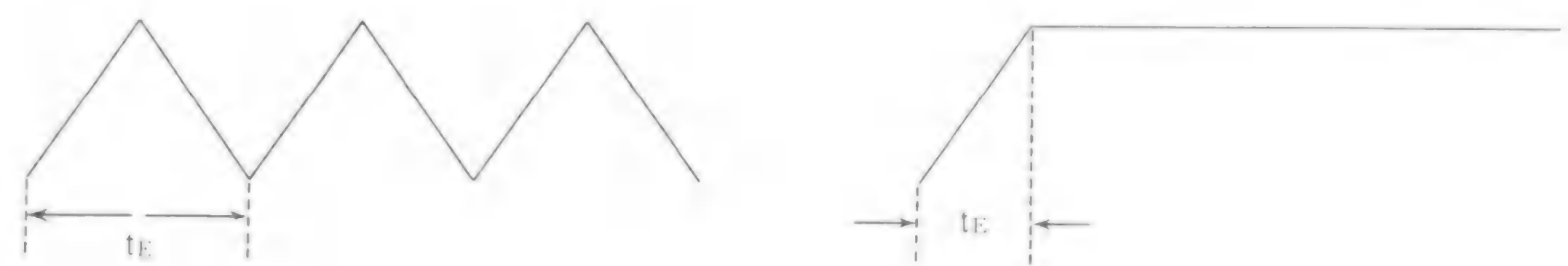
《エンベロープ・パターン》

R13の値	エンベロープ・パターン	音　の　例
0,1,2,3,9		爆発音、銃声、打撃音、ピアノ
4,5,6,7,15		蒸気吹出(シュッの音)
8		機関銃、踏切りのシグナル
10		ヘリコプター
11		
12		SLの音
13		管楽器、オルガン
14		波の音、サイレンの繰り返し

こうして、波形が決まると、次はその波の周期などの数量データを指定する必要があります。R 11, R 12でその指定を行ないます。

エンベロープ波形で、「音の鳴り始めから消滅するまで」あるいは「音の鳴り始めから音量最強になるまで」に要する時間を、**エンベロープの周期**とよびます。ここでは、 t_E で表わすことにします。 t_E が大きいほど、ゆったりした波形に、 t_E が小さいほどせわしない波形になります。 t_E の逆数 $1／t_E$ を考えて、**エンベロープ周波数**とよびます。これを f_E で表わします。

《エンベロープの周期》

t_E　　t_E

R 11、R 12 はエンベロープ周期 t_E の指定をします。R 11 は微調整値 FT、 R 12 は粗調整値 CT であって、これらの関係式は次の通りです。

$$t_E=\frac{1}{f_E},\quad f_E=\frac{fo}{256\times(CT\times256+FT)},\quad fo=2000000$$

たとえば、エンベロープ・パターン 4 番、CT=31、FT=255 と指定すると、$f_E=0.9538$(Hz)、$t_E=1.048$(sec) となりますから、エンベロープ波形は次のようになります。

1.048秒

```
SOUND 11, 255
SOUND 12, 31
SOUND 13, 4
```

3－8　SOUND 文の例

以上で、音出し関係の PSG のレジスタの説明は終えました。本節では、SOUND 文を利用した効果音の例を 2 つばかり挙げます。皆さんもいろいろ工夫してみて下さい。

《SOUND 文の例》

```
10 REM --- Gun Shot ---
20 '
30 SOUND 6,19
40 SOUND 7,7
50 SOUND 8,16
60 SOUND 9,16
70 SOUND 10,16
80 SOUND 12,15
90 SOUND 13,0
```

```
100 REM -- ﾌﾐｷﾘ --
110 '
120 SOUND 0,100
130 SOUND 1,0
140 SOUND 2,200
150 SOUND 3,0
160 SOUND 4,44
170 SOUND 6,0
180 SOUND 7,56
190 SOUND 8,16
200 SOUND 9,16
210 SOUND 10,16
220 SOUND 11,0
230 SOUND 12,20
240 SOUND 13,8
```

最近のパソコンの多くは、サウンド LSI を搭載していて、その中でも PSG(AY-3-8910) は最もポピュラーですから、X 1 以外のパソコンの本、雑誌記事も参考になります（他の機種とは、クロック周波数が異なることがあるので、音の間隔・周期などは若干異なりますが大筋は同じです)。たとえば、

工藤賢司著「マイコン・サウンドパック」(MSX POCKET BANK4)
(アスキー出版局刊)

などは実例に富んでいますから多いに参考になるでしょう。また、「サウンド・エディター」と称する音作り用のソフトも雑誌に発表されたり、市販されているようですから、これらも利用するとよいでしょう。

3－9 PSG関係のI／Oポート

本章はPSGの説明に重点を置きました。実際、SOUND文を用いて自分の望む音が出せるようになることが大切であって、マシン語は、ただそれを翻訳すればよいのです。では、遅ればせながら、マシン語による音出し法の解説をします。

PSGに関係して2つのI／Oアドレスが大切です。

《PSG関係のI／Oポート》

I/Oアドレス	機　　　能
1B00H〔注〕	PSGへのデータ出力（SOUND文の第2パラメータ相当）
1C00H〔注〕	PSGのレジスタ番号出力（SOUND文の第1パラメータ相当）

［注］これらのアドレス値は、上位8ビットのみ有効なので、たとえば、1B00H番地と1BFFH番地は同じ機能を持ちます。

次に挙げるサンプル・プログラムは、前節の「フミキリ」音をマシン語で出すものです。通常PSGへのデータ出力は、レジスタ番号とデータの対をメモリー上に格納しておいて、順にI／Oポートに出力する方法をとることが多いようです。

リスト　　マシン語による PSG control

```
100 REM *********************************
110 REM    ﾏｼﾝｺﾞ ﾆ ﾖﾙ PSG control
120 REM *********************************
130 '
140 CLEAR &HE000
150 GOSUB "ｶｷｺﾐ"
160 '
170 REM -- ﾃﾞｰﾀ ｾｯﾄ (ﾌﾐｷﾘ) ------------------------------------------
180 '
190 ADR=&HF000
200 POKE ADR,   0,100
210 POKE ADR+2, 1,0
220 POKE ADR+4, 2,200
230 POKE ADR+6, 3,0
240 POKE ADR+8, 4,44
250 POKE ADR+10,5,0
260 POKE ADR+12,6,0
270 POKE ADR+14,7,56
280 POKE ADR+16,8,16
290 POKE ADR+18,9,16
300 POKE ADR+20,10,16
310 POKE ADR+22,11,0
320 POKE ADR+24,12,20
330 POKE ADR+26,13,8
340 '
350 REM -- ﾒｲﾝ ------------------------------------------------------
360 '
370 CALL &HE000
380 '
390 END
400 '
410 LABEL "ｶｷｺﾐ" :REM ----------------------------------------------
420 '
430 ADR=&HE000
440 READ MC$
450 IF MC$="END" THEN RETURN
460 POKE ADR,VAL("&H"+MC$)
470 ADR=ADR+1
480 GOTO 440
490 '
```

```
500 REM -- ﾏｼﾝｺﾞ ﾙｰﾁﾝ --------------------------------------------------
510 '
520                       :REM  SNDDAT: EQU 0F000H
530                       :REM  ;
540                       :REM          ORG 0E000H
550                       :REM  ;
560 DATA 21,00,F0         :REM  SOUND:  LD  HL,SNDDAT   ;HL=data pointer
570 DATA 16,1C            :REM          LD  D,28        ;counter = 2*14
580 DATA 7E               :REM  LOOP:   LD  A,(HL)
590 DATA 01,00,1C         :REM          LD  BC,1C00H
600 DATA ED,79            :REM          OUT (C),A       ;PSG register select
610 DATA 05               :REM          DEC B           ;BC=1B00H
620 DATA 23               :REM          INC HL          ;data pointer inc
630 DATA 7E               :REM          LD  A,(HL)
640 DATA ED,79            :REM          OUT (C),A       ;data output to PSG
650 DATA 15               :REM          DEC D           ;counter dec
660 DATA 20,F2            :REM          JR  NZ,LOOP
670 DATA C9               :REM  EXIT:   RET
680                       :REM  ;
690 DATA END,             :REM  end mark
700                       :REM
710 REM -------------------------------------------------------------------
```

3-10 PSG内部の入出力ポート

PSGの第14、15レジスタは、入出力ポートにあてられています。

R 14＝入出力ポートA

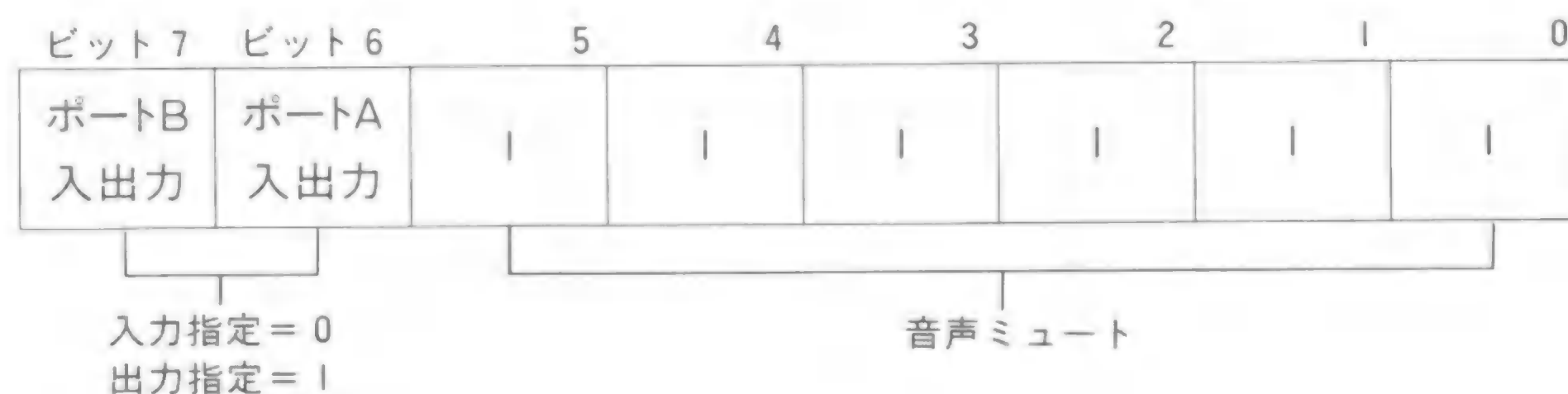
R 15＝入出力ポートB

これらのポートはいずれも、入力ポートとしても出力ポートとしても用いられますが、その指定はPSG第7レジスタR 7のビット7、6で行なわれます。

《R 7》

ビット7	ビット6	5	4	3	2	1	0
ポートB入出力	ポートA入出力	1	1	1	1	1	1

ビット7、6：入力指定＝0　出力指定＝1

ビット5～0：音声ミュート

《PSGのポートA、Bの入出力指定》

ポートA	ポートB	R7の設定値
入 力	入 力	SOUND 7、&H3F
出 力	入 力	SOUND 7、&H7F
入 力	出 力	SOUND 7、&HBF
出 力	出 力	SOUND 7、&HFF

さて、X１においては、R 14とR 15は２基のジョイスティック用ポートとして用いられます。また、デジタル・デロッパー接続時には、コンピュータからの制御用端子としても用いられます。

ジョイスティック使用時は、そのポートを入力ポートに指定しておく必要があります。たとえば、ジョイスティック１をポートＡに接続しておくとき、次のマシン語は、ジョイスティックの「トリガーボタン」をチェックするものです。

《マシン・コード》　　　　《ニーモニック》

```
01 00 1C      LD  BC, 1C00H
3E 0E         LD A, 14
ED 79         OUT (C), A
05            DEC B          ; BC=1B00H
ED 78         IN  A, (C)     ; A=ジョイスティック・コード
E6 20         AND 20H        ; ビット5(トリガー)を調べる
                             [トリガーON なら  Zフラグ=0]
                             [       OFF なら  Zフラグ=1]
```

第4章　汎用ポート8255

4－1　PPI 8255の概要

パソコンテレビX1には、VRAMの制御、内蔵カセットレコーダやプリンタとの交信、サブCPUとのデータのやりとり等を集中的に管理するために汎用のインターフェースLSIとして、２個の8255が搭載されています。X1におけるこれらのLSIの機能を理解することは、周辺機器をマシン語で制御する上でキーポイントといえるでしょう。そこで、8255というLSIの一般的機能から見てゆくことにします。

8255は、Programmable Peripheral Interface（PPIと略称される）ともよばれ、インテル社の8080CPUの周辺LSIとして開発されました。汎用のパラレル・インターフェースLSIでは最も多く使われているチップでしょう。

PPIは、**Aポート(PA)，Bポート(PB)，Cポート(PC)**とよばれる３つの８ビット入出力ポートと、それらのポートの使用法を決めるための８ビットの**コントロール・レジスタ**を持っています。ユーザーは、コントロール・レジスタに一定様式の命令(**コントロール・ワード**）を書き込むことで、8255を各種の入出力インターフェースとして利用することができる訳です。ここが、「プログラマブル」とよばれる所以です。

さて、CPUが、PPIの３つのポートと、コントロール・レジスタをアクセスするには、通常アドレス・バスの第１ビット（A_1）と、第０ビット（A_0）が用いられ、次のようになっています。

8555のポート，レジスタのアクセス

A_1	A_0	アクセスされるポート、レジスタ	Read/Write
0	0	ポートA(PA)	読み出し 書き込み ともに可能。
0	1	ポートB(PB)	
1	0	ポートC(PC)	
1	1	コントロール・レジスタ	書き込みのみ

X1において2個用いられている8255のうちの1つは、入出力命令（IN, OUT)時に、アドレス・バスの上位8ビットに1AHが指定されると、アクセス可能となり、アドレス・バスの下位2ビットで、ポート、レジスタのいずれかが選択されます。たとえば、

I／Oアドレス	1A00H＝8255	PA
	1A01H＝	PB
	1A02H＝	PC
	1A03H＝	コントロール・レジスタ

となっています。X1の場合、アドレス・バスの他のビットは未使用なので、たとえばI／Oアドレスの1A04H番地も8255PAを指定することになります（4の倍数の違いは無視される)。

次に、コントロール・レジスタに書き込むコントロール・ワードについて説明します。**コントロール・ワード**は、8ビットのデータからなりますが、その最上位ビット（MSB）が1か0かで意味が大きく異なります。

コントロール・ワード
- **MSB＝0のとき　ポートCの各ビットを独立にセット・リセットする。**
- **MSB＝1のとき　ポートのモード（動作の種類）を指定する。**

まず、PCのビットセット・リセット機能から見ましょう。

4-2　ポートCのビットセット・リセット

コントロール・レジスタに書き込むデータの様式は次のようになります（PC_nは、ポートCの第nビットを表わします)。

(MSB) ビット7	6	5	4	3	2	1	(LSB) ビット0
0	×	×	×				

- ビット7：ビットセット リセットを意味する
- ビット6～4：未使用（何でもよい）
- ビット3～1：PCのビット位置の指定
  - 000のとき PC_0
  - 001のとき PC_1
  - 010のとき PC_2
  - 011のとき PC_3
  - 100のとき PC_4
  - 101のとき PC_5
  - 110のとき PC_6
  - 111のとき PC_7
- ビット0：0のとき指定ビットをリセット　1のとき指定ビットをセット

データのうち第6ビット〜第4ビットは未使用なので何でもよいのですが、普通 000 にして用います。このとき、データの意味する所を一覧表にしたのが次図です。

ポートCのビットセット・リセット

コントロール・ワード	(16進例)	PC_7	PC_6	PC_5	PC_4	PC_3	PC_2	PC_1	PC_0
0×××0000	00H								0
0×××0001	01H								1
0×××0010	02H							0	
0×××0011	03H							1	
0×××0100	04H						0		
0×××0101	05H						1		
0×××0110	06H					0			
0×××0111	07H					1			
0×××1000	08H				0				
0×××1001	09H				1				
0×××1010	0AH			0					
0×××1011	0BH			1					
0×××1100	0CH		0						
0×××1101	0DH		1						
0×××1110	0EH	0							
0×××1111	0FH	1							

これらを理解すると、X1における次のようなプログラムの意味がわかります。

(例)　PC_6のビットセット

```
LD      A, 0DH          ;コントロール・ワード
LD      BC, 1A03H       ;1A03H=8255のコントロール・レジスタ
OUT     (C), A          ;8255 PC6のビットをセットする。
```

4−3　8255の3種類のモード

8255は、3種類の入出力モードを持っています。ここで、「モード」というのは、8255をどのような仕組の入出力インターフェースとして用いるかということです。

3種類のモードには、次の名称がつけられています。

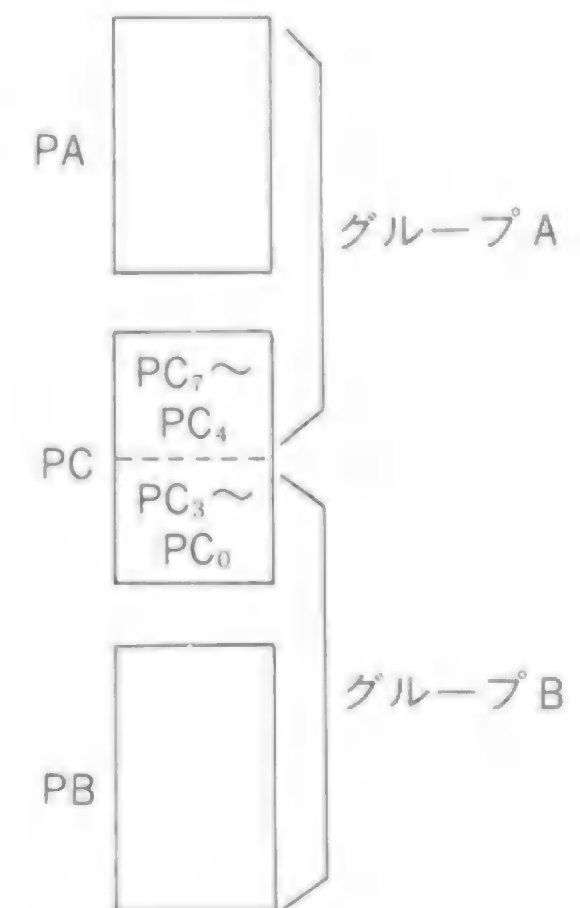
モード名称	インターフェースとしての働き
モード0	最も基本的なモードで、単なるI/Oポートとして動作する。
モード1	ストローブ付きI/Oポートとして動作する。
モード2	ストローブ付き双方向バスのI/Oポートとして動作する。

さて、8255には3つのポート（PA，PB，PC)があることはすでに述べましたが、これらは全く独立には用いることはできず、8255の動作モードにより決まった使い方をしなくてはなりません。

まず、3つのポートは大きく2つのグループに分けられることに注意しましょう。

グループA　…PAおよびPCの上位4ビット

グループB　…PBおよびPCの下位4ビット

PA　グループA
PC　PC_7〜PC_4　PC_3〜PC_0
PB　グループB

8255の動作モードの設定は、これらグループA，Bを対象として行なわれます。モード指定のコントロール・ワードはおおよそ次の様式になっています。

(MSB) ビット7　6　5　4　3　2　1　(LSB) ビット0

1

モード指定を意味する。

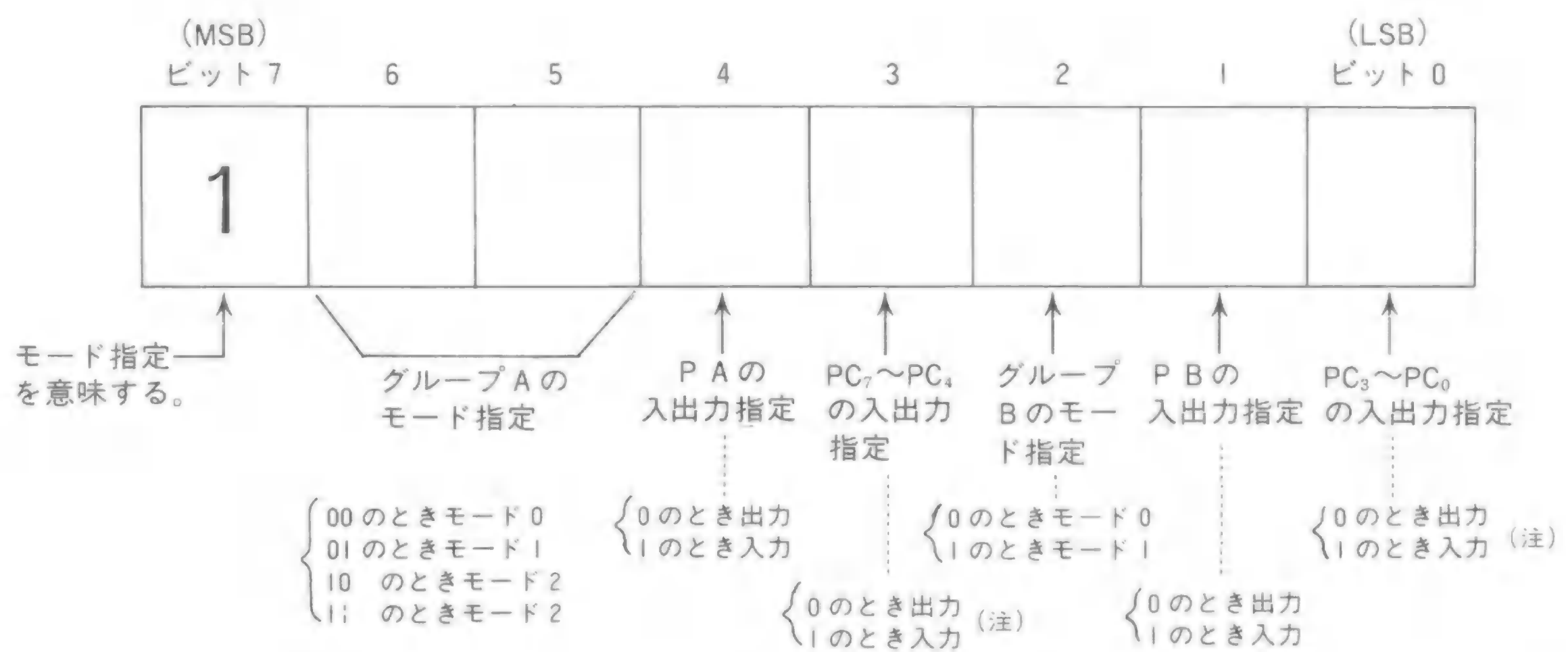
グループAのモード指定
- 00のときモード0
- 01のときモード1
- 10　のときモード2
- 11　のときモード2

PAの入出力指定
- 0のとき出力
- 1のとき入力

PC_7〜PC_4の入出力指定
- 0のとき出力
- 1のとき入力（注）

グループBのモード指定
- 0のときモード0
- 1のときモード1

PBの入出力指定
- 0のとき出力
- 1のとき入力

PC_3〜PC_0の入出力指定
- 0のとき出力
- 1のとき入力（注）

（注）PCの使われ方は、モード1，2では複雑。

では、各モードの動作を簡単に見てゆくことにします。8255について、さらに詳しく知りたい方は文献［10］，［12］，［14］，などを参照して下さい。

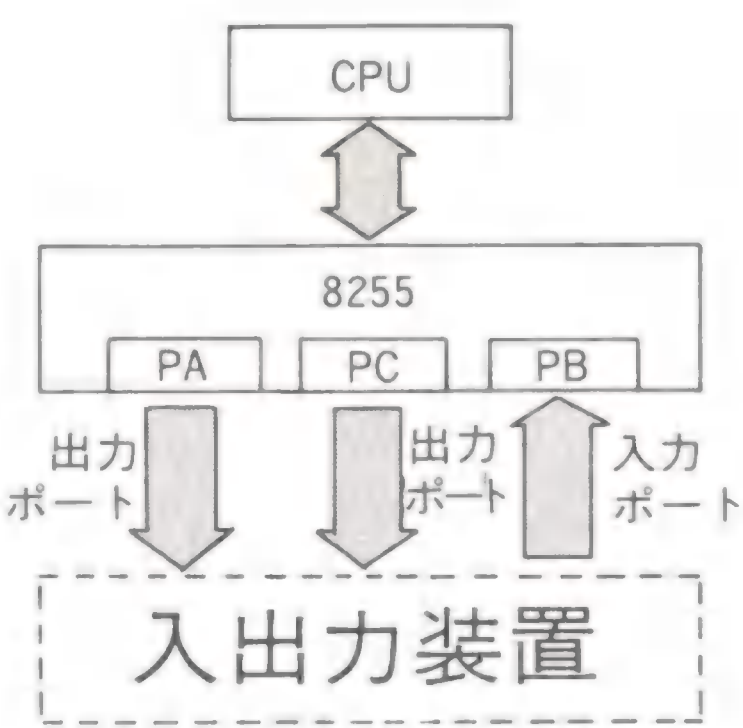
4−4　モード0の動作

モード0は最も単純な入出力ポートとして動作し、PA, PB, PC(上位4ビット)、PC(下位4ビット)という事実上4つのポートを各々独立に入力ポート、出力ポートとして16通りの指定をすることができます。実際に、コントロール・ワードとの対応を一覧表にすると次のようになります。

モード0設定用コントロール・ワード

コントロール・ワード		グループA		グループB	
MSB　　LSB	16進	PA	PC(上位4ビット)	PC(下位4ビット)	PB
1 0 0 0 0 0 0 0	8 0 H	出力	出力	出力	出力
1 0 0 0 0 0 0 1	8 1 H	出力	出力	入力	出力
1 0 0 0 0 0 1 0	8 2 H	出力	出力	出力	入力
1 0 0 0 0 0 1 1	8 3 H	出力	出力	入力	入力
1 0 0 0 1 0 0 0	8 8 H	出力	入力	出力	出力
1 0 0 0 1 0 0 1	8 9 H	出力	入力	入力	出力
1 0 0 0 1 0 1 0	8 A H	出力	入力	出力	入力
1 0 0 0 1 0 1 1	8 B H	出力	入力	入力	入力
1 0 0 1 0 0 0 0	9 0 H	入力	出力	出力	出力
1 0 0 1 0 0 0 1	9 1 H	入力	出力	入力	出力
1 0 0 1 0 0 1 0	9 2 H	入力	出力	出力	入力
1 0 0 1 0 0 1 1	9 3 H	入力	出力	入力	入力
1 0 0 1 1 0 0 0	9 8 H	入力	入力	出力	出力
1 0 0 1 1 0 0 1	9 9 H	入力	入力	入力	出力
1 0 0 1 1 0 1 0	9 A H	入力	入力	出力	入力
1 0 0 1 1 0 1 1	9 B H	入力	入力	入力	入力

たとえば、8255を次のようなインターフェースとして用いたいならば、コントロール・ワード82Hにより動作モード0を指定すればよいことになります。

CPU
8255
PA　PC　PB
出力ポート　出力ポート　入力ポート
入出力装置

[モード0指定の例]
コントロール・ワード
82H
による動作指定

4－5　ハンドシェーキング

モード１と２の動作を理解するには、「ハンドシェーキング」というデータの授受方法を知っておく必要があります。

まずたとえ話から。せっかちなＡさんと、のんびり屋のＢさんがチームを組んで仕事をする場面を想像して下さい。２人の間には仲介屋のＣさんがいて、相互連絡をするものとします。Ｃさんを介してのＡ，Ｂの仕事の手順は次のようになるでしょう。

手順１：　ＡさんからＣさんへ尋ねる。
「これからデータを送りますが、よいですか？」

手順２：　ＣさんからOKの返事があれば、ＡさんはＣさんへデータを伝える。

手順３：　同時に確認のため、「確かに送りましたよ」とＣさんへ伝える（ちょうど配達証明付の郵便のようなものです）。

手順４：　ＣさんはＡさんに「確かに受けとりましたよ」と返事する。

ここまでで、Ａさんの仕事の１ステップが完了した訳で、この件についての「責任」は仲介者のＣさんに移りました。Ａさんは別の仕事を始めることができます。

さて、今度は、ＣさんがＢさんに（Ａさんからの）データを伝える番ですね。

手順５：　ＣさんからＢさんに尋ねる。
「これからＡさんからのデータを送りたいのですが受け入れ準備はできていますか？」

手順６：　ＢさんからOKの返事があれば、ＣさんはＢさんへデータを送る。

手順７：　同時に、「確かに送りましたよ」とＢさんに伝える。

手順８：　ＢさんはＣさんに「確かに受けとりましたよ」と返事する。

手順９：　Ｃさんも仕事を終え、肩の荷が降りたので、Ａさんに「次の準備ができましたよ。」と伝える。

こうして、ＡさんからＢさんへのデータの伝達が１サイクル完了し、また手順１に戻ることになります。ＢさんからＡさんにデータを送るにも同様の手順を踏めばよい訳ですね。

Ａさん，Ｂさんを２つの装置に置きかえ、Ｃさんを装置間のインターフェースと考えると、これから扱いたい状況が納得できると思います。

２つの装置が別々のタイミングで動いているとき、それらは**非同期**であるといいます。非同期な装置間で、データを確実に送受したい時によく使われる方法が、たとえ話で述べたように相互に相手の状態を確認し合いながら段階的手順を踏んでデータの受け渡しを行なう方法です。これを**ハンドシェーキング**（hand shaking）とよびます。「握手」の意味ですね。２つの非同期な装置どうしが、ガッチリ握手したような状態をさしての用語なのでしょう。

ハンドシェーキングでは、相手の状態を確認するために、基本的に3つの信号を用います。

（イ）レディ（Ready）信号　＝「準備OK」を意味する。
（ロ）ストローブ（Strobe）信号　＝「確かに送ったよ」を意味する。［注］
（ハ）アクノリッジ（Acknowledge）信号＝「確かに受けとったよ」を意味する。

［注］　ストローブは、ストロボ・スコープなどとも使うように本来、閃光とか一瞬のパルス信号の意味ですが、データを送信した時の通知をするためのパルス信号の意味としてよく用いられているようです。

これらの用語を用いて、CPUとI／O装置とがインターフェースを介してハンドシェークする様子を図示してみましょう。

① CPU ←Readyの確認― インターフェース　I／O

② CPU ―Readyのとき データ→ インターフェース　I／O
ストローブ信号

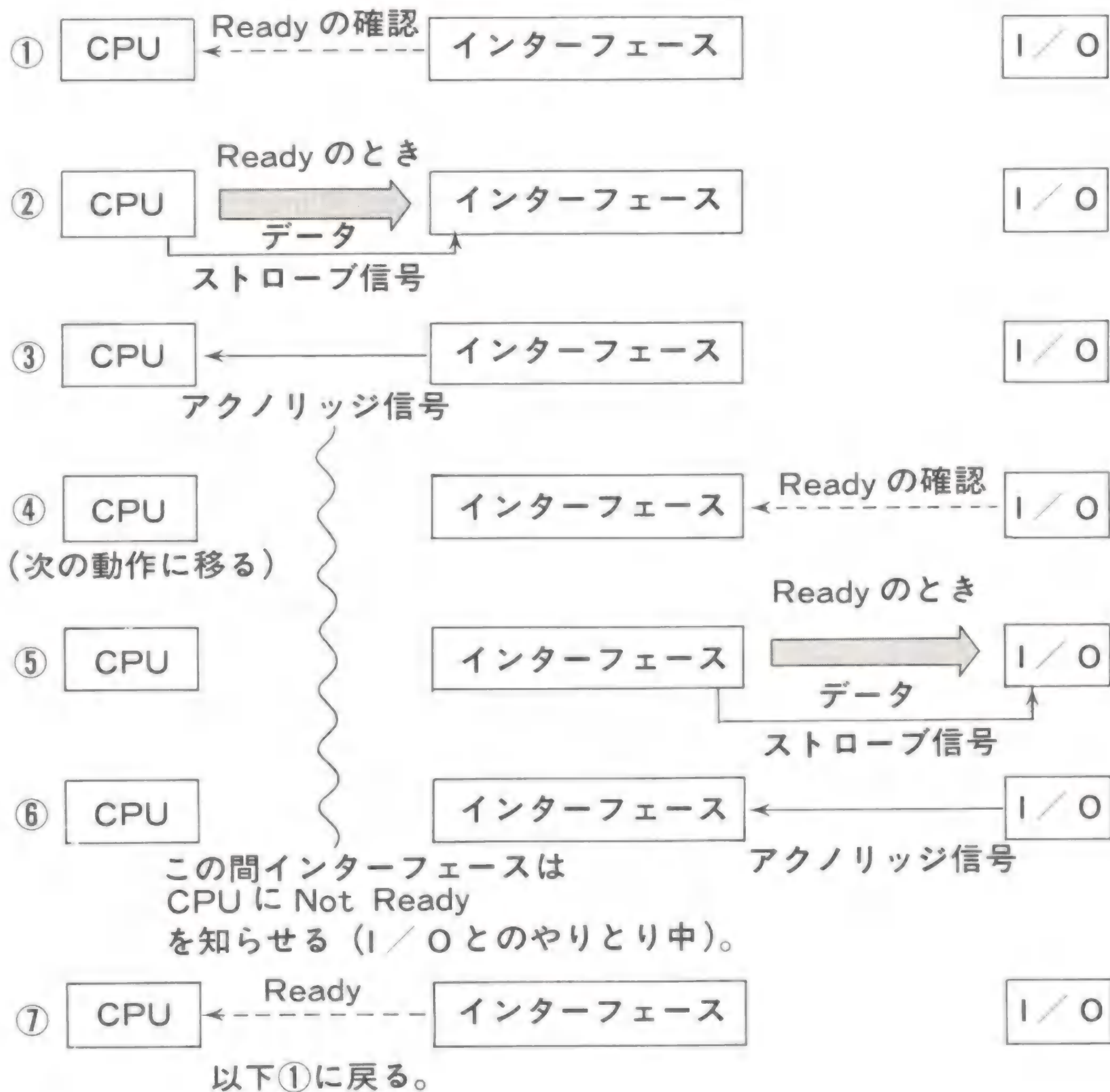
③ CPU ←― インターフェース　I／O
アクノリッジ信号

④ CPU　インターフェース ←Readyの確認― I／O
（次の動作に移る）

⑤ CPU　インターフェース ―Readyのとき データ→ I／O
ストローブ信号

⑥ CPU　インターフェース ←― I／O
アクノリッジ信号

この間インターフェースはCPUにNot Readyを知らせる（I／Oとのやりとり中）。

⑦ CPU ←Ready― インターフェース　I／O
以下①に戻る。

上図は、ハンドシェーキングによりCPUからI／O装置へデータを出力する場合ですが、I／OからCPUへデータが入力される時には、逆の流れになります。

4－6　モード1の動作

8255のモード1，モード2はともに、ハンドシェーキング付の入出力インターフェースとして動作させるモードです。まず、モード1から見てゆきましょう。

モード1は、グループAとグループBの各々に設定することができます。このとき、各グループは、8ビットのデータ用ポート(PAまたはPB)と、コントロール用ポート(PCの特定のビットがあてられる)とから構成されます。

モード1を入力ポート(I／O装置から8255へ)として使用すると、PCのいくつかの特定のビットは次の信号を扱うための端子となります。

$\overline{\text{STB}}$（ストローブ入力）
"L"レベル(0のこと)になると、I/O装置からのストローブ信号と判定して、I/O装置からのデータを受け入れる。

IBF（Input Buffer Full；入力バッファ・フル出力）
I/O装置からのデータを受けつけると、"H"レベル(1のこと)になり、I/O装置に「もう満杯だから次のデータを送るのは待て」と知らせる。CPU側がデータを取り込むと、再び"L"に戻り、I/O装置にReadyを知らせる。

INTR（Interrupt Request；割り込み要求出力）
8255がI/O装置から受けとったデータをもとに、CPUに割り込みをかけるために用いられる。

［注］　回路図などを見ているとよくありますが、端子や信号名の上に横線（バー）の引いてあるのは、負論理といって、それが"L"レベル（＝0）のときに意味を持つことを示しています。上で$\overline{\text{STB}}$がその例ですね。逆にバー付でないものは、正論理といって、"H"レベル（＝1）の時に意味を持ちます。

次に、モード1を出力ポート(8255からI／O装置へ)として使用する場合を見てみましょう。今度は、PCの特定のビットが次の信号の端子になります。

$\overline{\text{OBF}}$（Output Buffer Full；出力バッファ・フル出力）
CPUが出力命令によって8255のポートにデータを書き込むと"L"レベルになり、I/O装置側に「出力データ準備完了」を知らせる。I/O装置から、アクノリッジ信号を受けとると、"H"レベルに戻り「次のデータが準備できるまで待て」とI/O装置に知らせる。

$\overline{\text{ACK}}$（アクノリッジ入力）
I/O装置は、この端子を"L"レベルにすることで、8255に「データを受けとったよ」と知らせる。

INTR（Interrupt Request；割り込み要求出力）
I/O装置が、8255からデータを受けとったとき、CPUに割り込みをかけるために用いられる。

このように、PCのいくつかの特定のビットをコントロール用端子として、8255とI／O装置は、IBF，$\overline{\text{OBF}}$，$\overline{\text{STB}}$，$\overline{\text{ACK}}$などによりハンドシェーキングを行なうことができる訳です。

さて、次に、8255をモード1に設定するためのコントロール・ワードを述べましょう。グループA，Bのモードは独立に設定でき、グループA，グループBともにモード1である必要はないのですが、簡単のため次表では、グループA，Bを両方ともモード1で使用する為のコントロール・ワードを掲げます。PCの特定のビットの働きに注目して下さい。

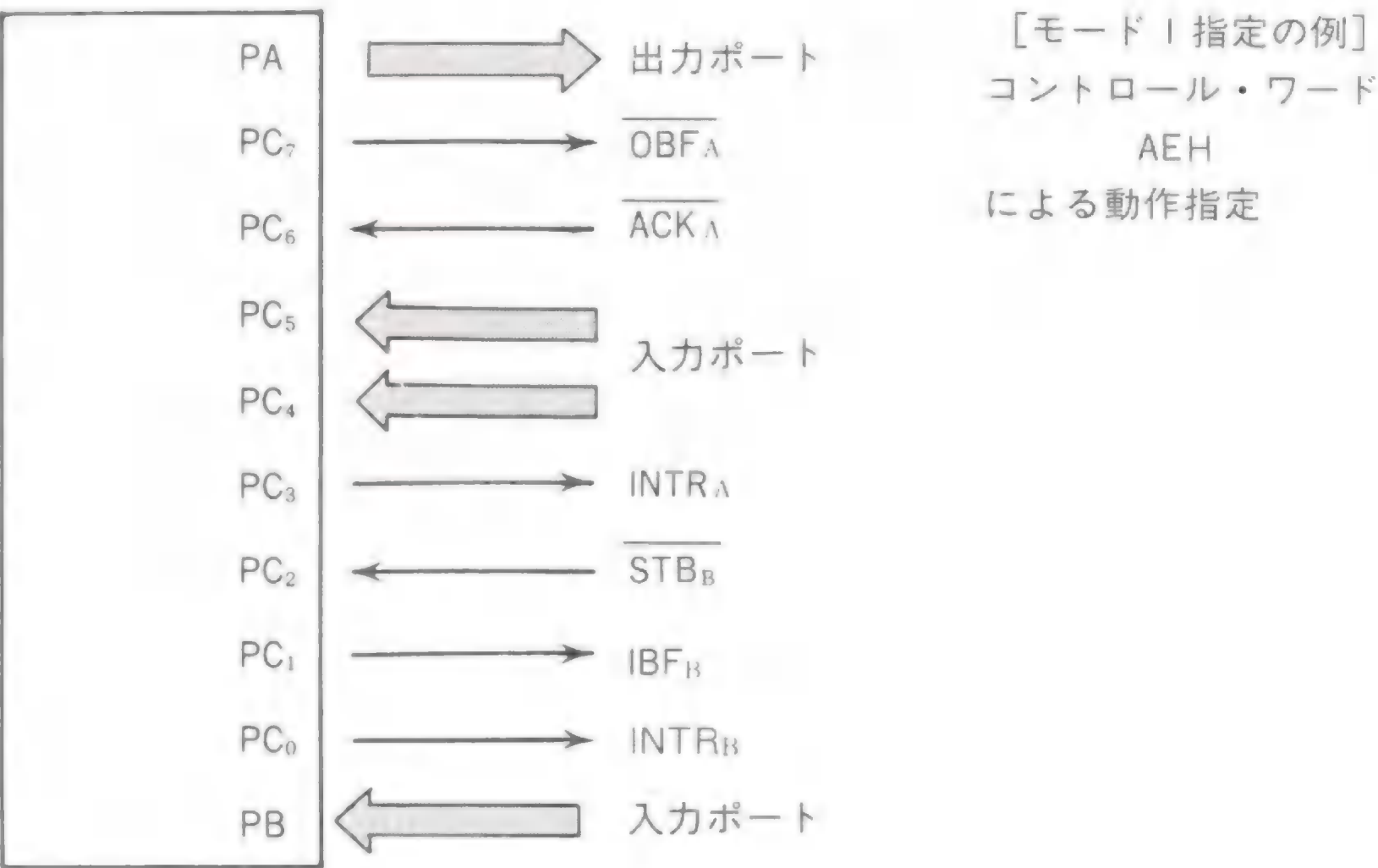
（モード1設定用コントロール・ワード）

コントロール・ワード		グループA						グループB			
MSB　　LSB	16進表示	ポートA	ポートC					ポートC			ポートB
			PC_7	PC_6	PC_5	PC_4	PC_3	PC_2	PC_1	PC_0	
1 0 1 0 0 1 0 X	A 4 H A 5 H	出　力	$\overline{OBF}_A$	$\overline{ACK}_A$	出　力		$INTR_A$	$\overline{ACK}_B$	$\overline{OBF}_B$	$INTR_B$	出　力
1 0 1 0 0 1 1 X	A 6 H A 7 H	出　力	$\overline{OBF}_A$	$\overline{ACK}_A$	出　力		$INTR_A$	$\overline{STB}_B$	IBF_B	$INTR_B$	入　力
1 0 1 0 1 1 0 X	A C H A D H	出　力	$\overline{OBF}_A$	$\overline{ACK}_A$	入　力		$INTR_A$	$\overline{ACK}_B$	$\overline{OBF}_B$	$INTR_B$	出　力
1 0 1 0 1 1 1 X	A E H A F H	出　力	$\overline{OBF}_A$	$\overline{ACK}_A$	入　力		$INTR_A$	$\overline{STB}_B$	IBF_B	$INTR_B$	入　力
1 0 1 1 0 1 0 X	B 4 H B 5 H	入　力	出　力		IBF_A	$\overline{STB}_A$	$INTR_A$	$\overline{ACK}_B$	$\overline{OBF}_B$	$INTR_B$	出　力
1 0 1 1 0 1 1 X	B 6 H B 7 H	入　力	出　力		IBF_A	$\overline{STB}_A$	$INTR_A$	$\overline{STB}_B$	IBF_B	$INTR_B$	入　力
1 0 1 1 1 1 0 X	B C H B D H	入　力	入　力		IBF_A	$\overline{STB}_A$	$INTR_A$	$\overline{ACK}_B$	$\overline{OBF}_B$	$INTR_B$	出　力
1 0 1 1 1 1 1 X	B E H B F H	入　力	入　力		IBF_A	$\overline{STB}_A$	$INTR_A$	$\overline{STB}_B$	IBF_B	$INTR_B$	入　力

［注］　表中、$\overline{OBF}_A$ ，$\overline{STB}_B$などとありますが、添字としてつけられているA，Bは、グループA，Bの区別を意味します。例えば$\overline{OBF}_A$は、グループAの$\overline{OBF}$端子を意味します。

たとえば、コントロール・ワードAEHを設定すると、グループA、Bともにモード1指定され、各端子は次のようになります。

PA → 出力ポート
PC_7 → $\overline{OBF}_A$
PC_6 ← $\overline{ACK}_A$
PC_5 ← 入力ポート
PC_4 ←
PC_3 → $INTR_A$
PC_2 ← $\overline{STB}_B$
PC_1 → IBF_B
PC_0 → $INTR_B$
PB ← 入力ポート

［モード1指定の例］
コントロール・ワード
AEH
による動作指定

4－7　モード2の動作

モード1では、ポートA、Bは入力ポート、出力ポートいずれかの働きしかしませんが、場合によっては、1つのポートを入出力の双方向ポートとして使用したいことがあります。このような機能はデータ・バスのような双方向バスとしての動作といえましょう。

モード2では、1個の8ビットバスを使って、入出力装置との情報を授受するための双方向バスの機能を持たせることができます。このモードは、グループAにのみ指定可能であって、8ビットの双方向バス機能を持つ入出力ポート（ポートA）と、ハンドシェーキング用の5ビット・コントロール・ポート（ポートCの上位5ビット）から構成されます。グループAをモード2に設定するとき、グループBはそれとは独立にモード設定ができ、モード0でも1でも構いません。

グループAをモード2で使用する時は、ポートAはある時は、モード1での入力ポートのように、またある時は、モード1での出力ポートのように動作します。従って、ポートCの上位5ビット（PC_7～PC_3）が受け持つハンドシェーキング用の制御信号は、モード1での入力、出力の制御信号を併せ持っており、次の5つからなります。

PC_7……$\overline{OBF_A}$　(Output Buffer Full)
PC_6……$\overline{ACK_A}$　(Acknowledge)
PC_5……IBF_A　(Input Buffer Full)
PC_4……$\overline{STB_A}$　(Strobe)
PC_3……$INTR_A$　(Interrupt Request)

これらの各制御信号の働きは、モード1の場合と同様なので、前節を参照して下さい。

さて次に、8255をモード2に設定するためのコントロール・ワードを掲げます。これらのコントロール・ワードの違いは、グループBのモード指定の違いによるものです。上の4つでは、グループBはモード0に、下の2つではグループBはモード1に指定されています。

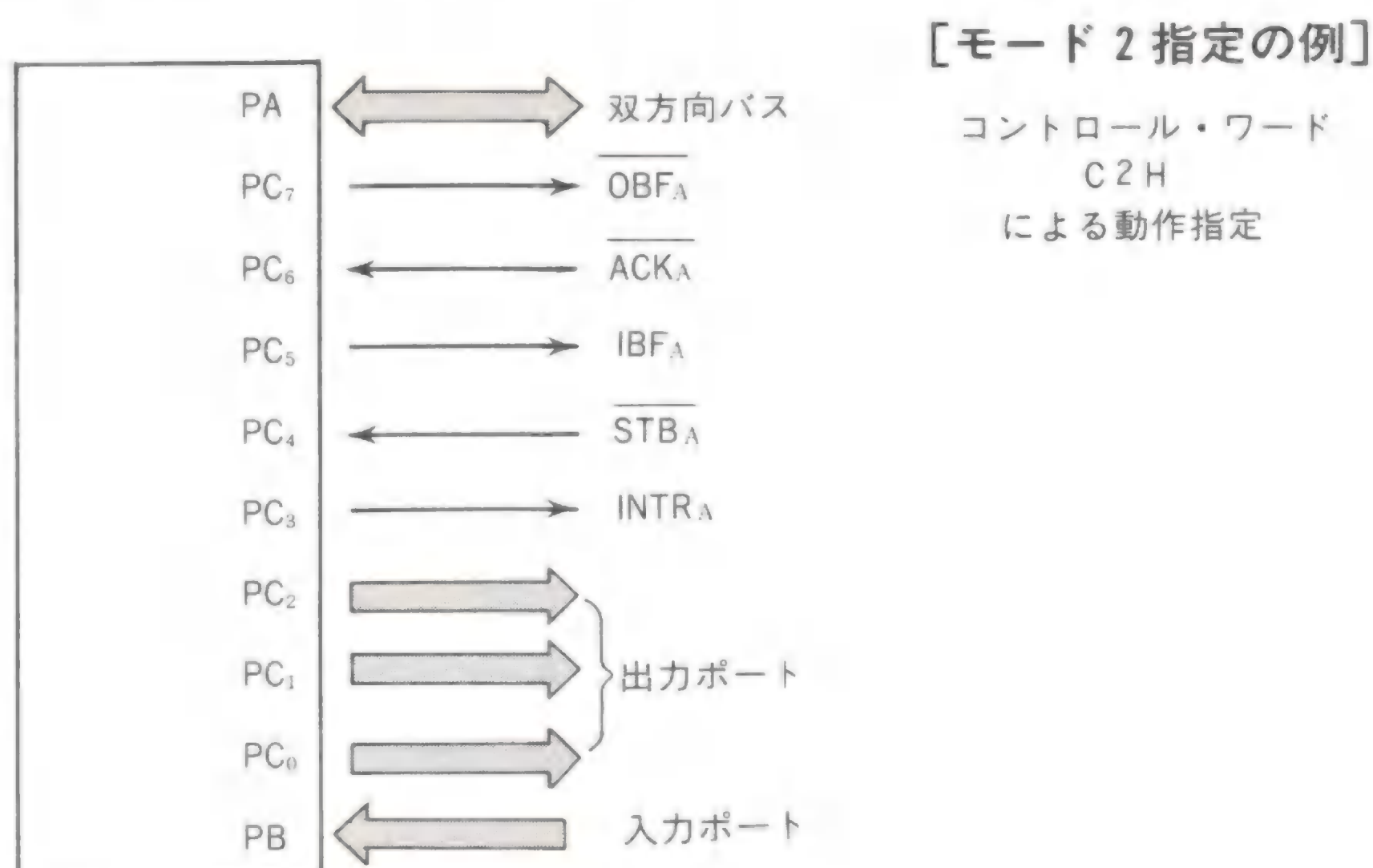
（モード2設定用コントロール・ワード）

コントロール・ワード		グループA						グループB			
MSB　LSB	16進(例)	ポートA	ポートC(上位)					ポートC(下位)			ポートB
			PC_7	PC_6	PC_5	PC_4	PC_3	PC_2	PC_1	PC_0	
11XXX000	C0H	双方向バス	$\overline{OBF_A}$	$\overline{ACK_A}$	IBF_A	$\overline{STB_A}$	$INTR_A$	出　力			出　力
11XXX001	C1H	双方向バス	$\overline{OBF_A}$	$\overline{ACK_A}$	IBF_A	$\overline{STB_A}$	$INTR_A$	入　力			出　力
11XXX010	C2H	双方向バス	$\overline{OBF_A}$	$\overline{ACK_A}$	IBF_A	$\overline{STB_A}$	$INTR_A$	出　力			入　力
11XXX011	C3H	双方向バス	$\overline{OBF_A}$	$\overline{ACK_A}$	IBF_A	$\overline{STB_A}$	$INTR_A$	入　力			入　力
11XXX10X	C4H	双方向バス	$\overline{OBF_A}$	$\overline{ACK_A}$	IBF_A	$\overline{STB_A}$	$INTR_A$	$\overline{ACK_B}$	$\overline{OBF_B}$	$INTR_B$	出　力
11XXX11X	C6H	双方向バス	$\overline{OBF_A}$	$\overline{ACK_A}$	IBF_A	$\overline{STB_A}$	$INTR_A$	$\overline{STB_B}$	IBF_B	$INTR_B$	入　力

たとえば、コントロール・ワードC2Hを設定すると、グループAがモード2、グループBがモード0に指定され、各端子は次のようになります。

[モード2指定の例]

コントロール・ワード
C2H
による動作指定

PA 双方向バス
PC7 → $\overline{OBF_A}$
PC6 ← $\overline{ACK_A}$
PC5 → IBF_A
PC4 ← $\overline{STB_A}$
PC3 → $INTR_A$
PC2
PC1 出力ポート
PC0
PB 入力ポート

4－8 X1内での8255

以上で、8255というLSIの一般的機能については、おわかりいただけたと思います。では、いよいよ、X1の中で8255がどのように使われているかを見てゆくことにしましょう。

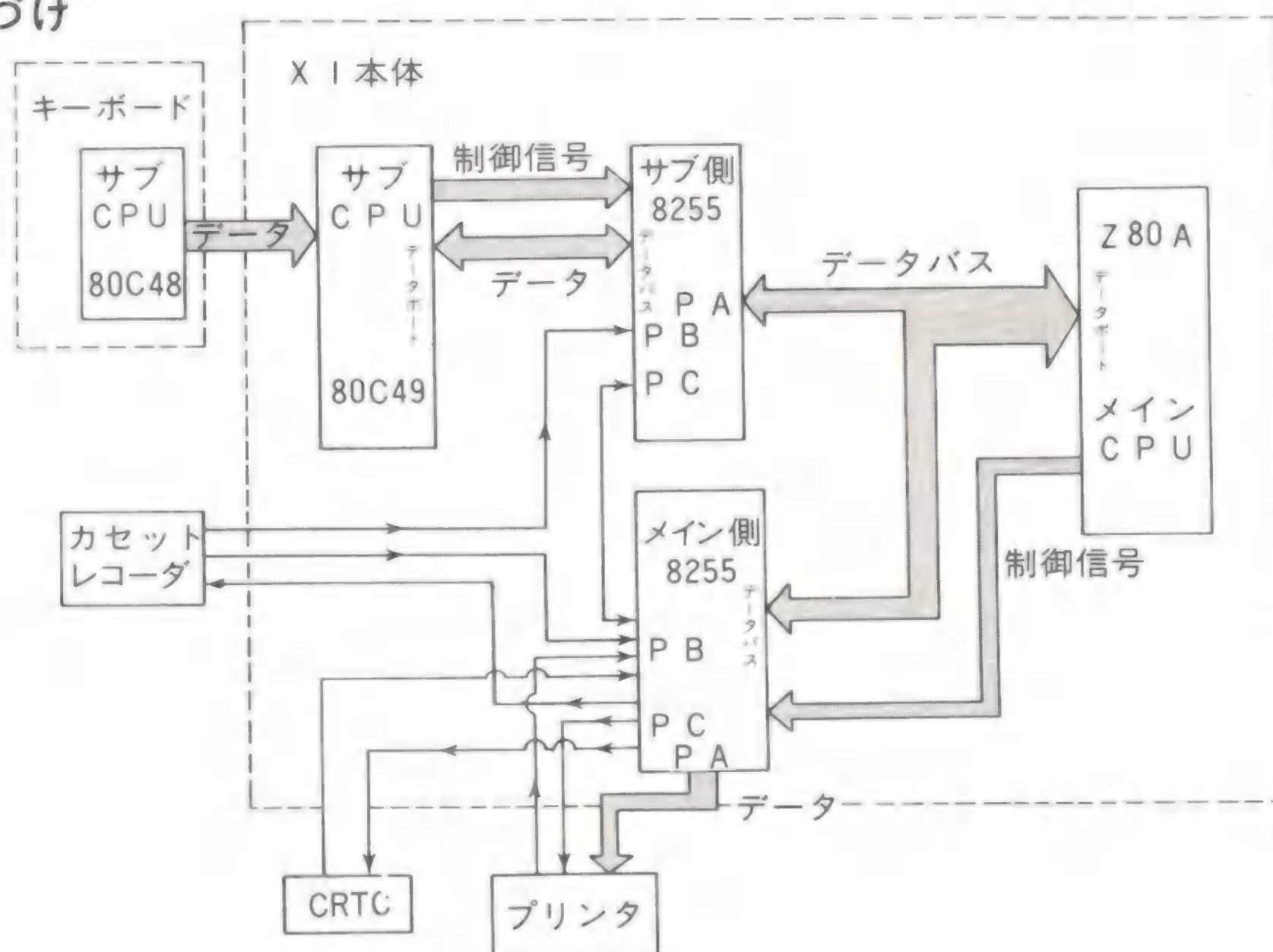
X1本体内部には、2個の8255が搭載されています。メーカーの型番号を示すと、

μPD 8255 AC	(日本電気製)
8255 C-5	(インテル製)

です。前者は、サブCPUとのインターフェースとして使われており、以下「サブ側8255」とよびます。また、後者は、メインCPUであるZ80A用のインターフェースとして働き、以下「メイン側8255」とよぶことにします。

これら2つの8255の大まかな位置づけは以下の通りです。

X1における8255の位置づけ

X1本体
キーボード
サブCPU 80C48
データ
サブCPU 80C49
制御信号
データ
サブ側8255 PA PB PC
データバス
Z80A メインCPU
カセットレコーダ
メイン側8255 PB PC PA
制御信号
データ
CRTC
プリンタ

4－9　サブ側 8255

サブ側 8255 のコントロール端子、データバス端子は、サブ CPU 80 C 49 に接続されていて、8255 のプログラミング(モード設定など)はサブ CPU に任されています。この部分は、解読不能であったため正確なことは不明なのですが、各ポートの使用のされ方から判断すると、次のように推測されます。

> 推測：　サブ側 8255 は、サブ CPU 80 C 49 から、コントロール・ワード C 2Hによって、グループ A はモード 2、グループ B はモード 0（PC_2〜PC_0は出力、PB は入力）に設定されている。

（注）　サブ側 8255 を、ユーザーがプログラミングすることはできません。

各端子の接続状態は次のようになっています。

サブ側 8255 の各端子

サブ CPU側データバス ⇔ D_7〜D_0

双方向バス PA_7〜PA_0 ⇔ メイン側データバス
($\overline{OBF_A}$) PC_7 → メイン側 8255 の PB_5へ
($\overline{ACK_A}$) PC_6 ← (I／O M I と RD より) [注]
(IBF_A) PC_5 → メイン側 8255 の PB_6へ
($\overline{STB_A}$) PC_4 ← (A←I／O (I A 0I H) で"L"となる)
($INTR_A$) PC_3 ------- 未使用
出力ポート PC_2 → カセット　発光ダイオード (WRITE) 点灯へ
出力ポート PC_1 → メイン側 8255 の PB_0へ (カセット走行フラグ)
出力ポート PC_0 → カセット EJECT へ
入力ポート PB_7 ← (PC_7の信号と同じもの)
入力ポート PB_6 ← (I／O M I と RD より)
入力ポート PB_5 ← カセット READ DATA
入力ポート PB_4 ← カセット EJECT スイッチ
入力ポート PB_3 ------- 未使用
入力ポート PB_2 ← カセット消去防止ツメ判定フラグ
入力ポート PB_1 ← カセットテープ装着フラグ
入力ポート PB_0 ← カセットテープ END フラグ

[注]（　）つきのものは、解析不十分の端子。

従って、サブ側 8255 の機能は、次のようにまとめられます。

（イ）　サブ CPU 80 C 49 とのデータ入出力のためのインターフェース
（ロ）　カセット・データ・レコーダをサブ CPU が制御するための入出力ポート

モード2に設定され、双方向バスとしての働きをしているポートAには、I／Oアドレス1900H番地が割りあてられているので、私たちはこのポートを通じて、サブCPUとのデータのやりとりを行なうことができます。ただしメインCPUとサブCPUとはクロック周波数が異なっているため、ハンドシェーキングによりデータ授受を行なう必要があり、ポートCの上位ビットPC_7〜PC_4は、そのための制御信号用端子です。とくに、出力バッファフル($\overline{OBF_A}$；PC_7)と入力バッファフル(IBF_A；PC_5)の両信号は大切で、これらは各々メイン側8255のポートBに出力されており、I／Oアドレス1A01H番地の第5、第6ビットを調べることで知ることができます。これらを用いたサブCPUとの交信の実際は、後述する第6章にて詳しく説明しますので、そちらを参照して下さい。

4-10　メイン側8255

メイン側8255は、メインCPUであるZ80Aの管理下にあるため、ユーザーがプログラムすることは可能です。ただし、すでに各端子が特定の周辺装置に接続されているために、ユーザーが勝手にモード設定をすると、正しい動作をしなくなるので注意が必要です。

メイン側8255のコントロール・レジスタは、I／Oアドレス1A03H番地に割りあてられています。(実際は、1A××H番地で下位2ビットが有効なため、1A03Hと4の倍数だけ離れた番地も同じ働きをします)。さて、X1のシステム・プログラム(HuBASICなど)では、起動後最初の「システム初期化ルーチン」中で、次の命令を実行しています。

```
LD      BC、1A03H
LD      A、82H
OUT     (C)、A
```

この部分こそが、メイン側8255のプログラミング(初期化)を行なうマシン語です。82Hは、コントロール・ワードで、8255のグループA、Bともにモード0に設定するものです。これにより、ポートA、Cが出力ポート、ポートBが入力ポートに指定されることになります（本章第4節のモード0の説明参照)。

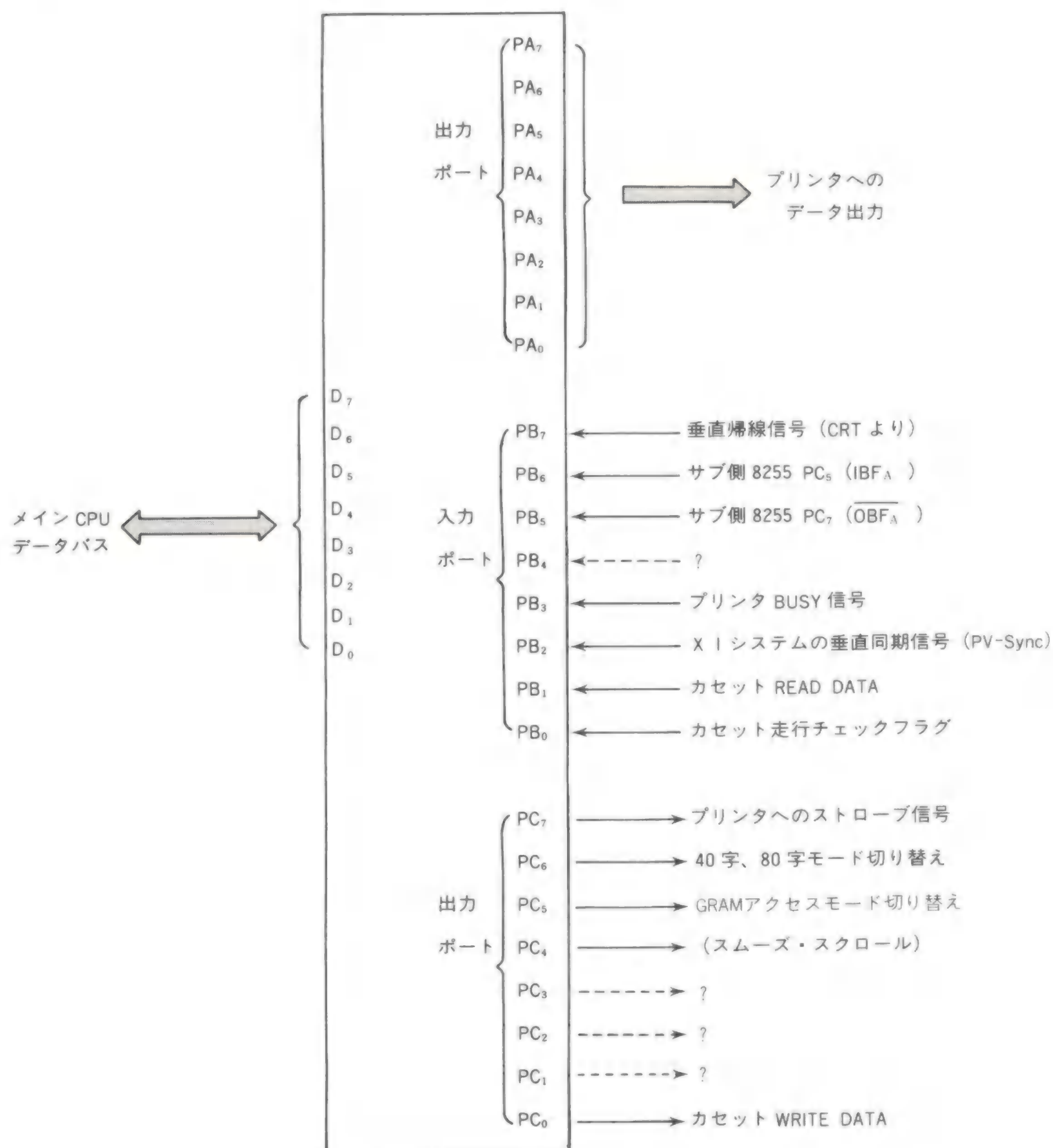
メイン側8255の各端子の接続状態は、次のようになっています。

メイン側8255の各端子

メイン CPU データバス ⇔ D_7 D_6 D_5 D_4 D_3 D_2 D_1 D_0

出力ポート
PA_7 PA_6 PA_5 PA_4 PA_3 PA_2 PA_1 PA_0 ⇒ プリンタへのデータ出力

入力ポート
PB_7 ← 垂直帰線信号（CRT より）
PB_6 ← サブ側 8255 PC_5（IBF_A）
PB_5 ← サブ側 8255 PC_7（$\overline{OBF_A}$）
PB_4 ← ?
PB_3 ← プリンタ BUSY 信号
PB_2 ← X I システムの垂直同期信号（PV-Sync）
PB_1 ← カセット READ DATA
PB_0 ← カセット走行チェックフラグ

出力ポート
PC_7 → プリンタへのストローブ信号
PC_6 → 40 字、80 字モード切り替え
PC_5 → GRAMアクセスモード切り替え
PC_4 → （スムーズ・スクロール）
PC_3 → ?
PC_2 → ?
PC_1 → ?
PC_0 → カセット WRITE DATA

4-11　ポートの使用例

8255の端子に接続されている周辺装置を見てもわかるように、とくにメイン側8255のポートの使い方は、X 1における入出力制御の要の1つと言えます(もう1つの要はサブCPU)。しかし、同時にわかりにくい部分でもあります。たとえば、次のマシン語プログラムは、グラフィック3画面の消去を高速に行ないますが(HuBASICのCLS 0に相当)、8255のポートの使われ方を理解すると意味が明瞭になります。

グラフィック画面オールクリア

```
CLS0:   F3       DI              ;disable interrupt
        01031A   LD   BC,1A03H
        3E0B     LD   A,0BH
        ED79     OUT  (C),A      ;8255 PC5 is '1'
        3D       DEC  A
        ED79     OUT  (C),A      ;8255 PC5 is '0'
;
        010040   LD   BC,4000H
LOOP:   AF       XOR  A
        0B       DEC  BC
        ED79     OUT  (C),A      ;I/O(BC) <-- 0
        78       LD   A,B
        B1       OR   C          ;BC = 0 ?
        20F8     JR   NZ,LOOP    ;if BC<>0 then LOOP
;
        0B       DEC  BC         ;BC = FFFFH
        ED78     IN   A,(C)      ;dummy
        FB       EI              ;inable interrupt
        C30010   JP   1000H      ;goto monitor
```

プログラム中、I／Oポートの1A03H番地にデータを出力する箇所があります。すでにおわかりのように、この番地はメイン側8255のコントロール・ポートにあてられていて、ここに0BH、0AHというデータを出力すると、ポートCのビットをセット・リセットします。今の場合は、

I／O（1A03H）← 0BH … PC_5のセット
I／O（1A03H）← 0AH … PC_5のリセット

となります。グラフィック画面の章でも触れたように、8255のCポート第5ビットにパルス信号を送ると、I／O空間のマップが変わって、GRAM同時アクセスモードとなります。このようにしておいてから、I／Oアドレスの3FFFH番地～0000H番地の内容をすべて00Hに変えるのが本プログラムの機能です。GRAM3画面のクリア終了後は、IN A、(C)を実行するとI／Oマップが戻ります。

このように、メイン側8255のポートの意味を知って初めて理解できる操作が沢山あります。本書でも以後、プリンタ制御（ポートA、B、C)、サブCPUとの交信（ポートB)、カセット・データ・レコーダ制御（ポートB、C）で8255のポートに触れますから、本章の内容をよく理解しておいて下さい。

第5章　プリンタ

5－1　なぜマシン語でプリンタを制御したいのか？

CRT画面をマシン語で高速制御することの良い点は、リアルタイム・アクションゲーム（インベーダーやゼビウスなど）などを引き合いに出せば、比較的納得しやすいと思います。では、プリンタをマシン語で制御することには、何か利益があるのでしょうか。

CRT画面と違って、プリンタの反応は遅く、従って、マシン語を利用しても、それほど時間が節約される訳ではありません。にもかかわらず、本章を敢えて「マシン語によるプリンタ制御」にあてたのには、２つの理由があります。

〔理由１〕

HuBASICには、画面のハード・コピーを行なうHCOPYという命令がありますが、たとえば、画面に絵を書いて、葉書大の小さな紙に印刷したい時に困るのです。HCOPYでは、横19㎝縦15㎝程の部分に印刷されてしまうのです。そこで私は一時期、これを縮小コピーにより小さくしたりしていたのですが、やはり、プリンタから葉書大の範囲に画面コピーを直接印刷したいと思うようになりました。HCOPYコマンドを使わずにハード・コピーするには、自分でハード・コピーのルーチンを作らねばなりません。それには当然のことに「マシン語によるプリンタ制御」ができなくてはなりません。

〔理由２〕

私は、X１というパソコン（＝私の愛機）をトータルに理解したいと考えています。HuBASICはすばらしいシステム・プログラムでX１の機能をよく引き出してくれます。しかし、X１の真の姿を自分で見るには、HuBASICのかなたにあるマシン語による入出力制御にまで迫らねばなりません。

LPRINTやHCOPYというプリンタ制御用のBASICコマンドを受けた時、X１はどのようにして、プリンタへデータを出力しているのでしょうか？そして、この問題に入っていくと、それが実は、前章で触れた「低速の入出力装置と高速のCPU間のハンドシェーキング」の格好の実例であることもわかりました。

こういった訳で、これから、プリンタをマシン語で制御する方法を説明します。私がシステムをそろえた頃は、X 1のプリンタといえば

CZ-800 P

という純正プリンタしかなかったのですが、最近は、シャープからも低価格の漢字プリンタが発売されたり、MZ 用のインクジェット・プリンタが利用できたり、また他のメーカーからもX 1対応のプリンタが出たりで、X 1で動くプリンタのラインアップも華やかになってきました。

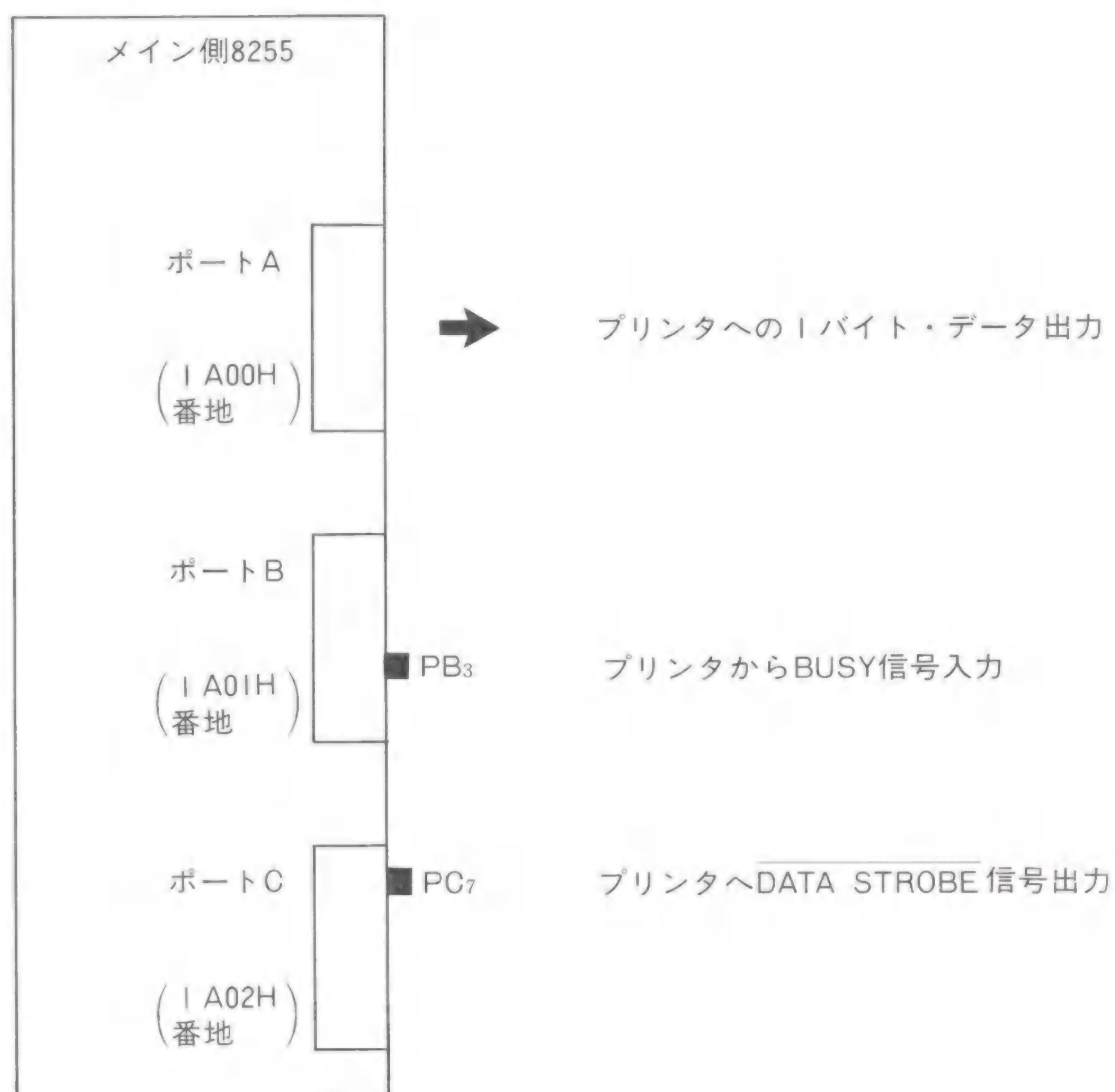
しかし、ここでは、最初のX 1用純正プリンタである CZ-800 P に的を絞って説明します。他のプリンタをお使いの方は、「プリンタ制御の原理」を読みとっていただき、出力コード等をそのプリンタ用に変更して下さい。

5－2 プリンタ関係I／Oポート

前章において、私たちは、インターフェース LSI である 8255 の機能と、その X 1内部での役割について学びました。

プリンタとの関係で必要事項をとり出すと次のようになります。

プリンタ関係のI／Oポート

メイン側8255		
ポートA（1A00H番地）		プリンタへの1バイト・データ出力
ポートB（1A01H番地）	PB_3	プリンタからBUSY信号入力
ポートC（1A02H番地）	PC_7	プリンタへ$\overline{\text{DATA STROBE}}$信号出力

5－3　プリンタとのハンドシェーキング

プリンタは、CPU に比べると、はるかに低速な出力装置ですから、CPU からプリンタへデータを送信して、確実に受けとってもらうには、前章でも触れたハンドシェーキングの方法をとる必要があります。

まず、プリンタが受信可能な状態にあるかどうかを、CPU は知らねばなりません。それを知らせるための端子が、I／O アドレス 1 A 01 H 番地の第 3 ビットです。この端子は、プリンタからの BUSY 信号を受けるものです。プリンタは受信可能な状態になった時、このビットを"L"(＝0) にすることで、CPU に READY を知らせます。

（注）BUSY 信号は正論理ですから、"H"レベル（＝1）である時に、 BUSY(忙しい) を示し、"L"レベル （＝0） である時に、 NOT　BUSY＝READY を意味します。

以下に掲げる簡単な BASIC プログラムを実行すると、I／O ポートの 1 A 0 H 番地と、プリンタからの BUSY 信号の様子を観察できます。

リスト　　PRINTER BUSY CHECK

```
100 REM *** PRINTER BUSY CHECK ***
110 '
120 INIT : WIDTH 40 : CLS
130 '
140 '
150 P=INP(&H1A01)
160 IF (P AND &B1000)=0 THEN M$="READY" ELSE M$="BUSY "
170 P$=RIGHT$("0000000"+BIN$(P),8)
180 LOCATE 0,0
190 PRINT "1A01H ﾊﾞﾝﾁ = ";P$
200 LOCATE 17,1
210 PRINT #0 CHR$(&H1E)
220 LOCATE 17,2 : IF M$="READY" THEN CREV 1
230 PRINT M$ : CREV 0
240 '
250 GOTO 150
```

X 1 とプリンタを配線してある状態（以下、当然この状態を想定します）で、プリンタの電源を ON にすると、第 3 ビットが 0 となり、画面に READY と表示されます。ここで、プリンタのセレクト・ランプ（SEL）を押して消し、プリンタをディセレクト（受信不能）状態にすると、第 3 ビットが 1 となり BUSY 表示になります。この状態で、LPRINT 等を実行すると、"Device offline"エラーとなる訳ですね。

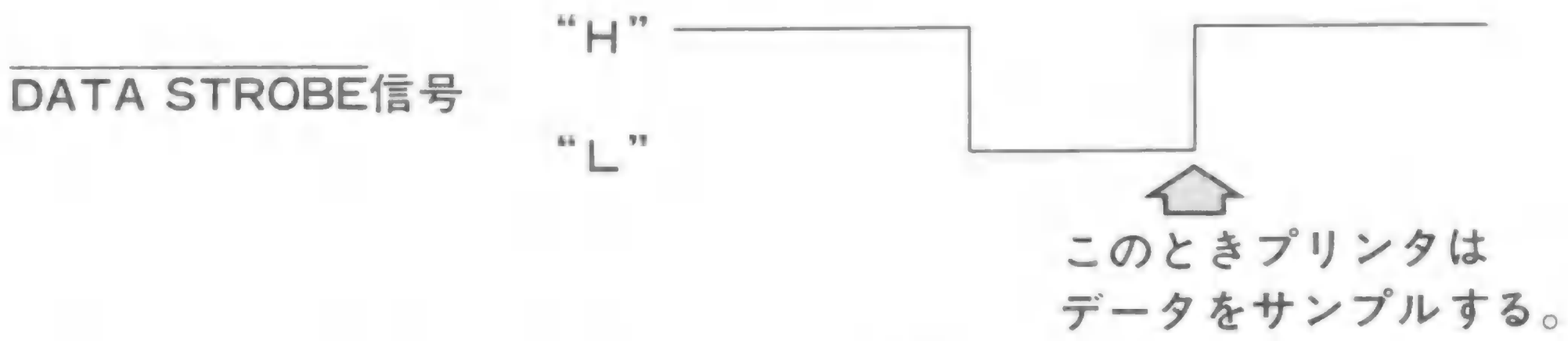
さて、面白いのは、プリンタ電源を OFF にした時です。筆者が実験した所、9 分 9 厘までは、BUSY 表示に変わったまま一定しているのですが、稀に、 BUSY → READY → BUSY と変化することがありました。この原因は、筆者にはよくわかりませんが、おそらく（コンデンサ等の）電気的な過渡現象が関係していると思われます。いずれにしても、プリンタ動作中にプリンタ電源を切ることは、しない方がよいようです。以上で、プリンタの BUSY 信号については、おわかりいただけたと思います。

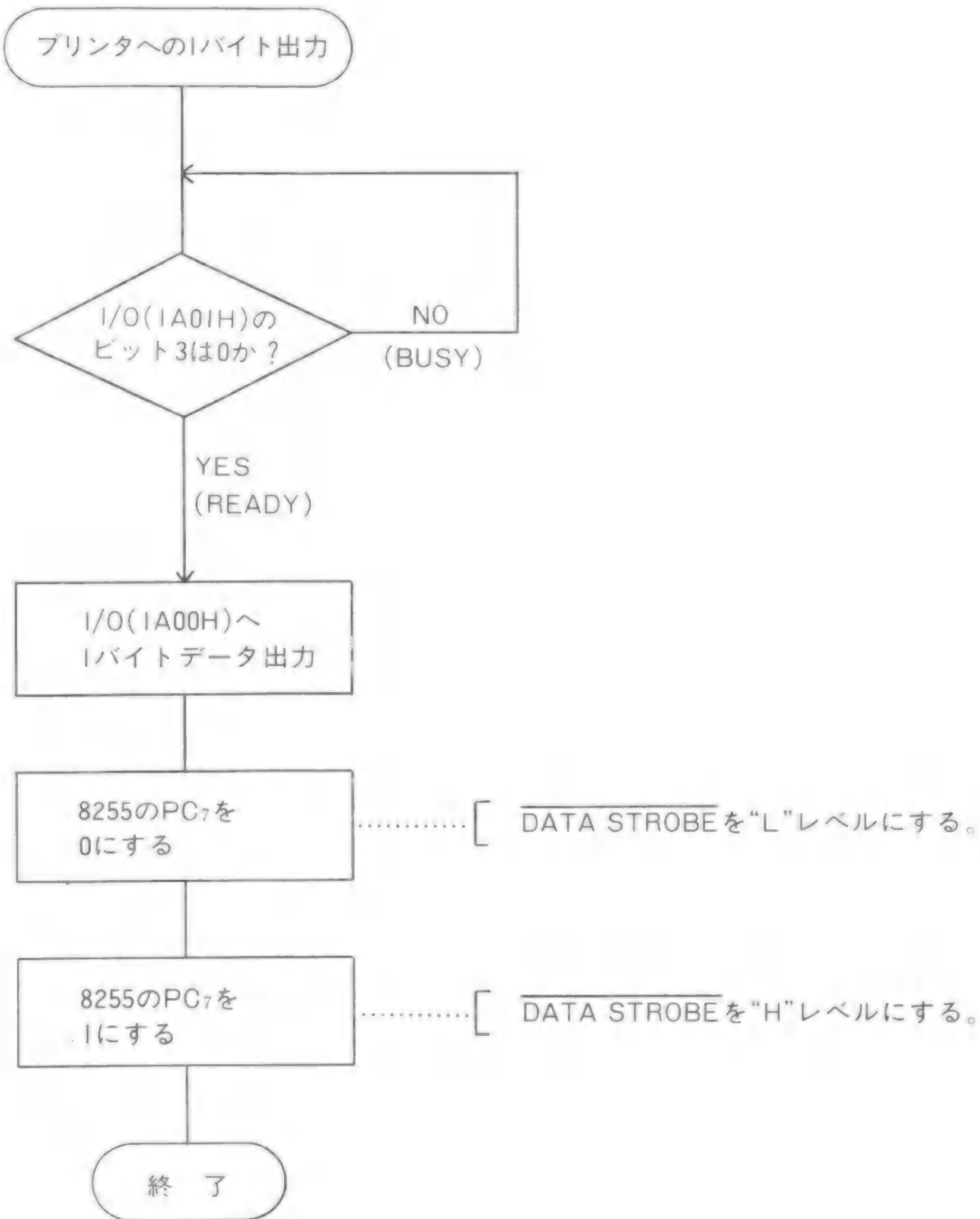
プリンタの受信準備完了を確認したら、CPU は、いよいよプリンタへデータを送ります。そのための出力ポートが、I／O アドレス 1 A 00 H 番地です。このポートに 1 バイトのデータを出力すれば、プリンタ側へ送られる訳です。

しかし、このままでは、「送りっぱなし」であって、プリンタには受けとってもらえません。そのためには、I／Oアドレス1A02H番地の第7ビットよりパルス信号を送る必要があります。この信号は、

$\overline{\text{DATA STROBE}}$

とよばれています（プリンタ CZ-800P の取扱説明書参照）。

上に棒がついていることからわかるように、負論理信号であって、プリンタ CZ-800P は、この信号が"L"レベルから"H"レベルに変わるときに、送られてきたデータをとり込む（サンプルする）ことになっています。

以上をまとめて、プリンタへの1バイト・データ出力の手続きを流れ図にしたものが下図です。

プリンタへの1バイト出力

5－4　プリンタへの1バイト出力ルーチン

前節で、プリンタと同期をとりながら(ハンドシェーキング)、1バイトデータを出力する仕組みがわかりましたから、早速、プログラムにしてみましょう。仕様は次の通り。

機　能	Aレジスタにセットされた1バイト・データを、プリンタ(CZ-800P)へ出力する。
留意点	①プログラムは、再配置可能(リロケータブル)なサブルーチンとする。 ②後に、文字列(メッセージ)出力ルーチンに組み込むことを考慮し、BC, DE, HLの各レジスタ対の内容は保存する。 ③プリンタOFFLINEを考慮し、BUSY信号が1のときは、一定時間待ち、それでもOFFなら、Hu BASIC内のモニター・ホットスタート1000番地へジャンプさせる(従って、他のモニターで実行する時は、ジャンプ先を変えて下さい)。 ④$\overline{\text{DATA STROBE}}$ パルスの出力については、8255のポートCビットセット・リセットモード(前章参照)を利用して行なう。

以下に掲げるサブルーチンは、E 000 H 番地から格納してありますが、上で述べたように「リロケータブル」ですから、必要ならば別の番地に転送してお使い下さい。

リスト　Subroutine of lprint one byte

```
0000                    ;*****************************************
0001                    ;*                                       *
0002                    ;*      Subroutine of lprint one byte    *
0003                    ;*                                       *
0004                    ;*             by Y.Shimizu              *
0005                    ;*                                       *
0006                    ;*             1984.5.5 FRI              *
0007                    ;*                                       *
0008                    ;*****************************************
0009                    ;
0010                    ;
0011                    ;===== input condition =============
0012                    ;
0013                    ;       A : 1 byte data for lprint
0014                    ;
0015                    ;===== registers preserved =========
0016                    ;
0017                    ;       BC,DE,HL
0018                    ;
0019                    ;===================================
0020                    ;
0021                    ;
0022 1000               MONHOT:EQU  01000H          ;Hu-monitor hot start
0023                    ;
0024                    ;
0025                           ORG  0E000H
0026                    ;
0027                    ;
0028                    ;***** registers push *****
0029                    ;
0030                    ;
0031 E000 C5                   PUSH BC
0032 E001 D5                   PUSH DE
0033 E002 E5                   PUSH HL
0034                    ;
0035 E003 F5                   PUSH AF              ;preserve data
0036                    ;
0037                    ;
0038                    ;***** busy-checking *****
0039                    ;
0040                    ;
0041 E004 1610          BUSYCK:LD   D,10H           ;waiting counter initialize
0042 E006 210000               LD   HL,0000H
0043                    ;
0044 E009 01011A               LD   BC,1A01H        ;I/O adrs of Port B of 8255
0045                    ;
0046 E00C ED78          LOOP1: IN   A,(C)
0047 E00E E608                 AND  08H             ;checking of BUSY (8255 PB3)
```

```
0048 E010 280B          JR   Z,PROUT       ;if printer is ready then lprint
0049                 ;
0050 E012 2B            DEC  HL            ;waiting routine
0051 E013 7D            LD   A,L
0052 E014 B4            OR   H
0053 E015 20F5          JR   NZ,LOOP1
0054 E017 15            DEC  D
0055 E018 20F2          JR   NZ,LOOP1
0056                 ;
0057 E01A C30010     ERROR: JP   MONHOT        ;case of device offline
0058                 ;
0059                 ;
0060                 ;***** output routine *****
0061                 ;
0062                 ;
0063 E01D F1         PROUT: POP  AF            ;A reg = data for lprint
0064                 ;
0065 E01E 01001A        LD   BC,1A00H      ;I/O adrs of Port A of 8255
0066 E021 ED79          OUT  (C),A         ;data output to printer
0067                 ;
0068 E023 01031A        LD   BC,1A03H      ;I/O adrs of control port of 8255
0069 E026 3E0E          LD   A,0EH
0070 E028 ED79          OUT  (C),A         ;strobe "L" level
0071 E02A 3E0F          LD   A,0FH
0072 E02C ED79          OUT  (C),A         ;strobe "H" level (data sampling)
0073                 ;
0074                 ;
0075                 ;***** exit *****
0076                 ;
0077                 ;
0078 E02E E1         EXIT:  POP  HL            ;registers pop
0079 E02F D1            POP  DE
0080 E030 C1            POP  BC
0081                 ;
0082 E031 C9            RET                ;exit of routine
0083                 ;
0084 E032               END
```

このリストは、あくまで読者の皆さんが、X 1 でのプリンタへの出力法を理解する際の参考として掲げました。もし、HuBASIC ないしは、そのモニターを基礎にプログラムを実行することがわかっているならば、HuBASIC のモニター（以後 **Hu モニター** とよびます）内に、すでに完成されたサブルーチンが組み込まれているので、これを利用すると便利です。

《Hu モニター・システム・サブルーチン》

名　称	プリンタへの1文字出力。
アドレス	12E2H番地
機　能	Aレジスタの内容を、プリンタへ出力する。
レジスタ	A, BC, DE, HLの各レジスタの内容は保存される。
注　意	プリンタがOFFの時は、Device offlineエラー表示へ、ジャンプする。

5－5　プリンタへの文字列出力

前節で紹介した Hu モニター内のシステム・サブルーチン 12E2H を利用して、文字列（メッセージ）をプリンタへ出力するプログラムを作ってみましょう。以下のプログラムは、 DATA とラベルをはったアドレスから格納されるデータ列をプリンタへ連続して出力するものです。データ列の終了マークは、00 H に設定してあります。

リスト　LPRINT SAMPLE PROGRAM

```
0000                    ;****************************************
0001                    ;*                                      *
0002                    ;*        LPRINT SAMPLE PROGRAM         *
0003                    ;*                                      *
0004                    ;*            by Y.Shimizu              *
0005                    ;*                                      *
0006                    ;*            1984.5.5 FRI              *
0007                    ;*                                      *
0008                    ;****************************************
0009                    ;
0010                    ;
0011                    ;====  Hu monitor  system subroutine  ====
0012                    ;
0013 12E2               LPTOUT:EQU   12E2H           ;LPRINT 1 byte
0014 1000               MONHOT:EQU   1000H           ;Hu monitor hot start
0015                    ;
0016                    ;----------------------------------------
0017                    ;
0018                    ;
0019                    ;
0020                           ORG   0E000H
0021                    ;
0022                    ;
0023 E000 210FE0        LPRINT:LD    HL,DATA         ;HL = data top adrs
0024                    ;
0025 E003 7E            LOOP:  LD    A,(HL)          ;A  = data for LPRINT
0026 E004 FE00                 CP    0
0027 E006 CA0010               JP    Z,MONHOT        ;if END-mark then MONHOT
0028 E009 CDE212               CALL  LPTOUT          ;call sub. of LPRINT 1 byte
0029 E00C 23                   INC   HL              ;data adrs increment
0030 E00D 18F4                 JR    LOOP
0031                    ;
0032                    ;
0033                    ;
0034 E00F 0D0A          DATA:  DB    0DH,0AH         ;CR and LF (Let new line)
0035 E011 54686973             DB    'This is a sample program '
     E015 20697320
     E019 61207361
     E01D 6D706C65
     E021 2070726F
     E025 6772616D
     E029 20
0036 E02A 0D                   DB    0DH             ;CR (carriage return)
0037 E02B 666F7220             DB    'for printer control.'
     E02F 7072696E
     E033 74657220
     E037 636F6E74
     E03B 726F6C2E
0038 E03F 0D0A                 DB    0DH,0AH         ;CR and LF (start pr-out)
0039 E041 00                   DB    00H             ;END-mark of message (=00H)
0040                    ;
0041                    ;
0042 E042                      END
```

5－6　プリンタの制御コード

プリンタへ出力するコードの中には、文字に対応するものと、プリンタ動作を指定する制御コードの2種類があります。前節で登場した0DH, 0AHという2つのコードは制御コードです。

シンボル（呼　称）	コード	機　　　　能
CR (Carriage Return)	0DH	(データ列)＋0DHの形で用いて、データ列を印字後、プリンタのヘッドを左端に戻す（このとき、ヘッドは新しい行の左端になる)。
LF (Line Feed)	0AH	(データ列)＋0AHの形で用いて、データ列を印字後、1行改行して、ヘッドを左端に戻す。

（注）両者よく似ていますが、特徴的な違いは、例えば、前にデータ列がなく、単に0DHだけ、あるいは単に0AHだけ受信した時に起こります。前者では、すでにヘッドが左端にあるので、何も起こらず、後者では1行改行が生じます。

以上の2つのコードは、多くのプリンタで共通のようですが、これから述べる制御コードは、プリンタによって違いがありますから、別のプリンタをお使いの際は、そのマニュアルを参照して下さい。ここでは、CZ-800Pの制御コードを述べます。

プリンタに細かな指示を与えるためのコードの多くは、**エスケープ・シークエンス**（escape sequence）といって、ESCコードに続く、データ列の形をとります。ESC（escape；**エスケープ**）とは、ASCIIコード　1BH（10進で27）に割りあてられているコードで、X1では画面表示される文字には対応しません。それでは何のためにあるかというと、以下に続く符号の意味を変える働きをします。たとえば、1BH，41Hというデータ列は、画面表示すると、単に文字Aとなりますが、このようにエスケープ・シークエンスとして用いると特別な意味に変えることができます。この場合プリンタCZ-800Pの制御コードとしては、1行の桁数を長く設定する命令に対応します。

エスケープ・シークエンスのうち、本章で以後用いる2つの命令をここでは採り上げます。（残りについては、プリンタの取扱説明書21ページを御覧下さい）。

エスケープ・シークエンス	バイト数	機　　　能
1BH，25H，39H，n_3	4	改行間隔をn_3/144インチに設定する。ただし、n_3＝00Hのときは、もとの1/6インチあるいは1/8インチ改行幅に戻す。
1BH，25H，32H，n_1，n_2	5	この命令を受けつけると、以後、(n_1×256＋n_2)バイト分を、ビット・パターンとしてプリンタ・ヘッドに出力する（グラフィック印字）。

（注）表中、n_1，n_2，n_3は各々、ユーザーが与える1バイト数値データです。たとえば、1BH，25H，39H，0CHというエスケープ・シークエンスは、12／144＝1／12インチの改行指示を意味します。

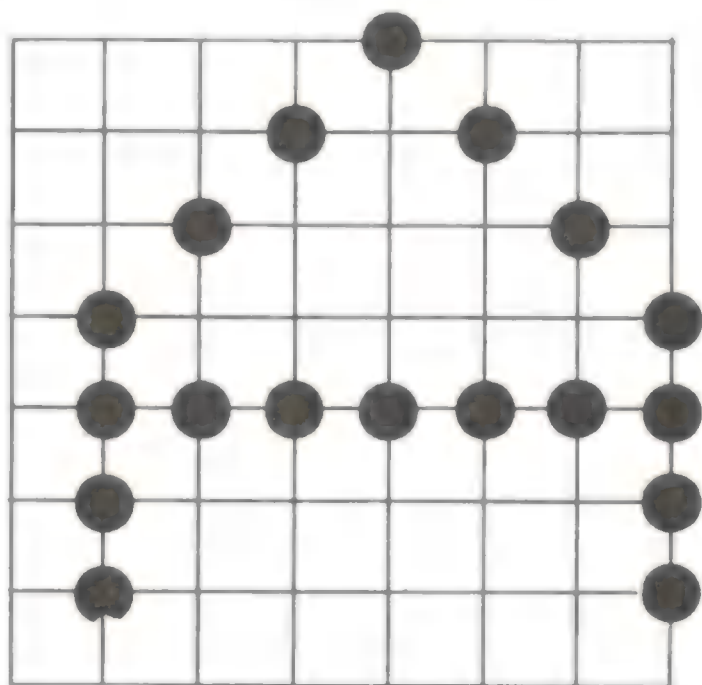
前者の命令は、プログラム・リスト等をとる時に、行の間隔をつめて印字したい場合に用いると便利です。私はよく、n_3＝12Hとして改行間隔を設定します。

後者のグラフィック印字命令は、難しいので次節で改めて説明します。

5－7　グラフィック印字

通常のキャラクタ（キャラクタ・コード 20～FFH）は、CRT 画面に表示されるドット・パターンと同様のものが、プリンタ内部のメモリー内に保存されているので、「文字列出力」の所で述べたようにキャラクタ・コードを出力すれば、プリンタは自動的に印字してくれます。

しかし、自分で作成した文字パターンや、ハード・コピーする時の画面のパターン等は、プリンタ内に用意されていないので、自分でプリンタのヘッドを操作して、印字しなくてはなりません。これを可能にするのが、**グラフィック印字機能**です。

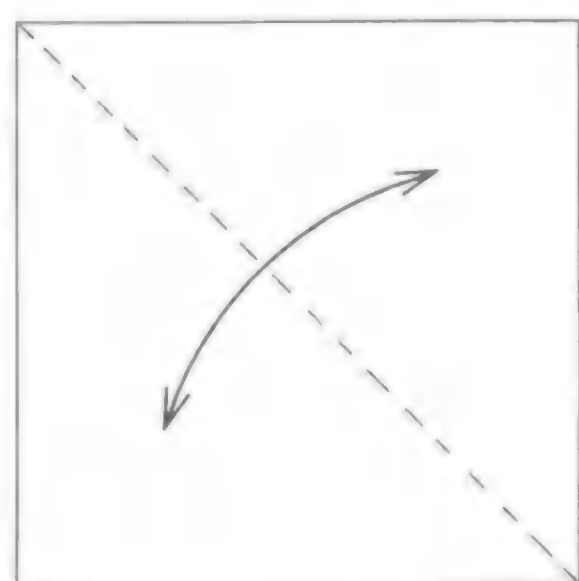
CZ-800 P のようなドット・インパクト方式のプリンタは、ヘッドからワイヤとよばれる突起が出て、インクリボン紙に押しつけ、ワイヤの跡を残すことにより、印字を行ないます。 CZ-800 P の場合、ワイヤは縦に 8 個並んでいて、ヘッドが左から右に移動するに従って、縦 1 列ずつのパターンを紙に打ってゆきます。たとえば、文字Aのドットパターンを印字するには、

図のようなパターンを縦に見て、左から右へ 1 列ずつ打っていく訳です。

さて、注意すべきなのは、上のようなドット・パターンを印字するのに、プリンタへどのようなデータ列を与えればよいか？　という点です。プリンタの取扱説明書 16 ページによると、 CZ-800 P においては、ヘッドとデータの対応は次のようになります。

ヘッドピン1（最上段）	↔	データ第7ビット（MSB）
2	↔	6
3	↔	5
4	↔	4
5	↔	3
6	↔	2
7	↔	1
8	↔	0　（LSB）

従って、先程の文字Aパターンは、

	ビットパターン		16進表示
第1列データ（最も左の縦1列）	0 0 0 0 0 0 0 0	=	0 0 H
第2列データ	0 0 0 1 1 1 1 0	=	1 E H
第3列データ	0 0 1 0 1 0 0 0	=	2 8 H
第4列データ	0 1 0 0 1 0 0 0	=	4 8 H
第5列データ	1 0 0 0 1 0 0 0	=	8 8 H
第6列データ	0 1 0 0 1 0 0 0	=	4 8 H
第7列データ	0 0 1 0 1 0 0 0	=	2 8 H
第8列データ（最も右の縦1列）	0 0 0 1 1 1 1 0	=	1 E H

という8バイトのデータ列に対応します。大切な点は、ヘッドに与えるデータ列を上のように並べると、印字パターンと「転置」(transpose) の関係になっていることです。

「転置」とは上図のように、正方形状に並んだパターンを、その左上から右下への対角線について折り返す操作のことです。

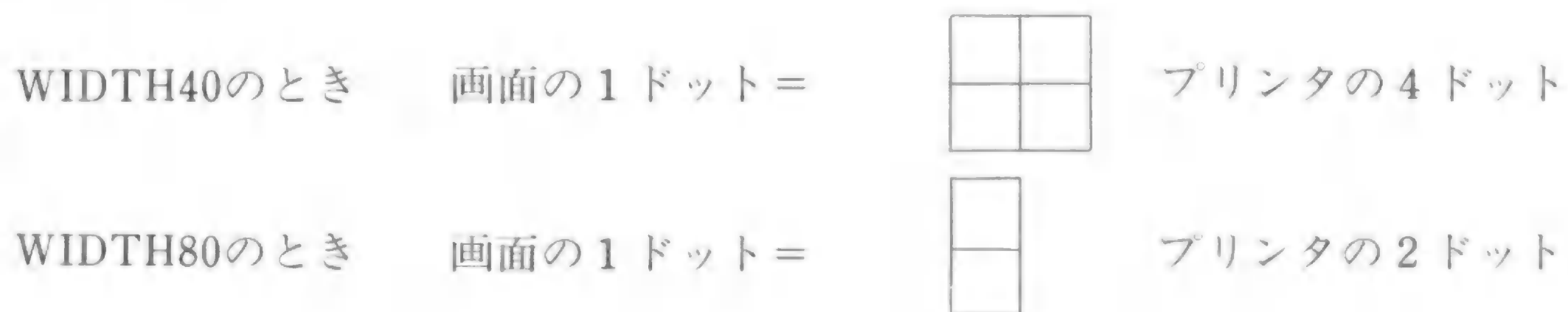
私たちは、メモリー上に、ビット・パターンを構成してから、プリンタに出力しますが、上述の理由からそれを一度「転置」してからでないと、印字された時に引っくり返ってしまいますから注意しましょう。

以上の注意の下に、グラフィック印字指定後に、文字Aのビットパターンを出力するBASICプログラムを以下に掲げます。

```
100 REM ﾁﾁﾁﾁ ｸﾞﾗﾌｨｯｸ ｲﾝｼﾞ ﾁﾁﾁﾁ
110 '
120 REM -- ｸﾞﾗﾌｨｯｸ ｲﾝｼﾞ ｼﾃｲ 8 ﾊﾞｲﾄ --
130 '
140 LPRINT CHR$(&H1B,&H25,&H32,0,8);
150 '
160 REM -- A ﾉ ﾋﾞｯﾄ ﾊﾟﾀｰﾝ --
170 '
180 LPRINT CHR$(0,&H1E,&H28,&H48,&H88,&H48,&H28,&H1E)
190 '
200 END
```

5－8　画面ハード・コピー

グラフィック印字機能を理解すると、その応用として、画面ハード・コピーのルーチンを自作することができます。本章の冒頭で述べたように、HuBASICでサポートしているHCOPY コマンドでは、不満な場合があります。HuBASIC内で HCOPY を処理するルーチンを解析すると、

WIDTH40のとき	画面の1ドット＝	⊞	プリンタの4ドット
WIDTH80のとき	画面の1ドット＝	⊟	プリンタの2ドット

に対応して、グラフィック印字パターンを出力していることがわかります。確かに、このようにしておけば、WIDTH 40 でも 80 でも、ハードコピーの縦横比をほぼ1対1に設定することが可能です。

しかし、葉書大のような小さい範囲に画面を美しくコピーするには、やはり、（画面の1ドット）＝（プリンタの1ドット）　という対応があるのが望ましいと思われます。

以上のような考えで作成したのが、これから紹介する「画面ハードコピー」のサブルーチンです。リロケータブルではありせんが、メモリーの後ろの方、FD 00 H 番地から配置しておきましたので、HuBASICと共存させることは十分に可能でしょう。ただし、画面とプリンタの1ドットを各々対応させたために、 WIDTH 40 では正常にコピーされますが、 WIDTH 80 の画面は、縦横比が1：2と横長になります。これは仕方のないことでしょう。これはこれで、テキスト画面のハードコピーについては利用価値があると思います。WIDTH 80 で縦横比をそろえたハードコピーは、HuBASICの HCOPY で行なって下さい。

メモリー・マップは次の通りです。

自作ハードコピー・サブルーチン
メモリー・マップ

FD00H～FD97H	ハードコピー・ルーチン本体
FDA 0 H～FDF 4 H	テキスト画面ハードコピーのサブルーチン(HCOPY相当)
FE00H～FE 2 DH	グラフィック3画面ハードコピーのサブルーチン(HCOPY0相当)
FE30H～FE 5 EH	VRAMアドレス計算サブルーチン
FE60H～FE 7 BH	ビット・パターン転置サブルーチン
FE80H～FEA I H	ワーキング・エリア

実行に先立って、FD 56 H 番地～FD 5 BH 番地の 6 バイトに、使用サブルーチンを書き込んで下さい。初期状態は、テキスト画面のみのハード・コピー指定にしてありますが、次のようにするとグラフィック画面のコピーがとれます。

FD56H～FD58H	FD59H～FD5BH	ハード・コピーのモード
CD, A0, FD	00, 00, 00	テキスト画面のみ（HCOPY）
CD, 00, FE	00, 00, 00	グラフィック画面のみ（HCOPY 0）
CD, A0, FD	CD, 00, FE	テキスト及びグラフィック画面(HCOPY 4)

HuBASIC のモニターから ＊G FD 00 で起動するか、HuBASIC より CALL &HFD 00 により実行して下さい。特に、WIDTH 40 では、美しいハード・コピーが得られることでしょう。

以下に、「ソース・リスト」，「チェック・サム表」，40 桁表示画面でのハード・コピー例、80 桁表示画面でのハード・コピー例を挙げておきます。

ソースリスト

```
0000                    ;*************************
0001                    ;
0002                    ;  ｶﾞﾒﾝ ﾊｰﾄﾞ ｺﾋﾟｰ
0003                    ;
0004                    ;   by  Y.Shimizu
0005                    ;
0006                    ;   1984.6.10-6.14
0007                    ;
0008                    ;*************************
0009                    ;
0010                    ;
0011                    ;
0012                    ;=== Hu monitor subroutine ===
0013                    ;
0014 12E2               LPTOUT:EQU  12E2H           ;LPRINT A-reg.
0015 0033               PCGRD: EQU  0033H           ;READ from PCG
0016                    ;
0017                    ;=== Hu monitor working area ===
0018                    ;
0019 0036               BRKWK: EQU  0036H           ;BREAK key work
0020 0007               WIDTH: EQU  0007H           ;WIDTH value (40 or 80)
0021 00E9               SCRPG: EQU  00E9H           ;VRAM screen page (0 or 4)
0022                    ;
0023                    ;
0024                    ;
0025                           ORG  0FD00H
0026                    ;
0027                    ;--------------------------
0028                    ;     HCOPY main routine
0029                    ;--------------------------
0030                    ;
0031 FD00 21A1FE               LD   HL,YLOC
0032 FD03 3600                 LD   (HL),0          ;Y=0
0033                    ;
0034 FD05 3E1B                 LD   A,27            ;LPRINT CHR$(27,37,57,N3)
0035 FD07 CDE212               CALL LPTOUT          ;(N3)/144 ｲﾝﾁ ｶｲｷﾞｮｳ
0036 FD0A 3E25                 LD   A,37
0037 FD0C CDE212               CALL LPTOUT
0038 FD0F 3E39                 LD   A,57
0039 FD11 CDE212               CALL LPTOUT
0040 FD14 3E0F                 LD   A,15            ;N3=15  (5/48 ｲﾝﾁ ｶｲｷﾞｮｳ ｼﾃｲ)
0041 FD16 CDE212               CALL LPTOUT
0042                    ;
0043 FD19 21A0FE        LOOPY: LD   HL,XLOC
0044 FD1C 3600                 LD   (HL),0          ;X=0
0045                    ;
0046 FD1E 3A3600       BREAK: LD   A,(BRKWK)
0047 FD21 FE03                 CP   3
0048 FD23 CA84FD               JP   Z,EXIT          ;if BREAK then EXIT
0049                    ;
0050 FD26 3A0700        BYTE:  LD   A,(WIDTH)       ;WIDTH ﾉ ﾁｪｯｸ
0051 FD29 FE50                 CP   80
```

```
0052 FD2B 2805              JR    Z,BT640        ;WIDTH 80 ﾅﾗ BT640 ﾍ
0053                 ;
0054 FD2D 214001     BT320: LD    HL,320         ;ﾌﾟﾘﾝﾀ ｼｭﾂﾘｮｸ 320 ﾊﾞｲﾄ (40 ｼﾞ ﾌﾞﾝ)
0055 FD30 1803              JR    GRAPH
0056                 ;
0057 FD32 218002     BT640: LD    HL,640         ;ﾌﾟﾘﾝﾀ ｼｭﾂﾘｮｸ 640 ﾊﾞｲﾄ (80 ｼﾞ ﾌﾞﾝ)
0058                 ;
0059 FD35 3E1B       GRAPH: LD    A,27           ;LPRINT CHR$(27,37,50,N1,N2)
0060 FD37 CDE212            CALL  LPTOUT         ;ｸﾞﾗﾌｨｯｸ ｲﾝｼﾞ ( N1*256+N2 ) ﾊﾞｲﾄ
0061 FD3A 3E25              LD    A,37
0062 FD3C CDE212            CALL  LPTOUT
0063 FD3F 3E32              LD    A,50
0064 FD41 CDE212            CALL  LPTOUT
0065 FD44 7C                LD    A,H            ;A=N1 (ﾊﾞｲﾄ ｽｳ ｼﾞｮｳｲ 8 ﾋﾞｯﾄ)
0066 FD45 CDE212            CALL  LPTOUT
0067 FD48 7D                LD    A,L            ;A=N2 (ﾊﾞｲﾄ ｽｳ  ｶｲ 8 ﾋﾞｯﾄ)
0068 FD49 CDE212            CALL  LPTOUT
0069                 ;
0070 FD4C 2180FE     LOOPX: LD    HL,BITPAT      ;ﾜｰｸ ｴﾘｱ ｸﾘｱ
0071 FD4F AF                XOR   A
0072 FD50 0608              LD    B,8
0073 FD52 77         LP1:   LD    (HL),A
0074 FD53 23                INC   HL
0075 FD54 10FC              DJNZ  LP1
0076                 ;
0077 FD56 CDA0FD            CALL  PATST1         ;bit pattern set sub.
0078 FD59 000000            DB    00H,00H,00H    ;HCOPY4 ﾉ ﾀﾒﾉ ｽﾍﾟｰｽ
0079                 ;
0080 FD5C 2180FE            LD    HL,BITPAT
0081 FD5F 0608              LD    B,8
0082 FD61 7E         LP2:   LD    A,(HL)         ;ﾋﾞｯﾄ ﾊﾟﾀｰﾝ ﾌﾟﾘﾝﾀ ｼｭﾂﾘｮｸ
0083 FD62 CDE212            CALL  LPTOUT
0084 FD65 23                INC   HL
0085 FD66 10F9              DJNZ  LP2
0086                 ;
0087 FD68 3A0700            LD    A,(WIDTH)
0088 FD6B 47                LD    B,A            ;B = WIDTH ﾉ ｱﾀｲ (40 or 80)
0089                 ;
0090 FD6C 21A0FE            LD    HL,XLOC        ;X=X+1
0091 FD6F 34                INC   (HL)
0092 FD70 7E                LD    A,(HL)
0093 FD71 B8                CP    B              ;WIDTH ﾄ ｸﾗﾍﾞﾙ
0094 FD72 DA4CFD            JP    C,LOOPX        ;if X < (WIDTH) then LOOPX
0095                 ;
0096 FD75 3E0A              LD    A,10           ;LPRINT CHR$(10)
0097 FD77 CDE212            CALL  LPTOUT         ;ﾗｲﾝ ﾌｨｰﾄﾞ (ｶｲｷﾞｮｳ)
0098                 ;
0099 FD7A 21A1FE            LD    HL,YLOC        ;Y=Y+1
0100 FD7D 34                INC   (HL)
0101 FD7E 7E                LD    A,(HL)
0102 FD7F FE19              CP    25
0103 FD81 DA19FD            JP    C,LOOPY        ;if Y<25 then LOOPY
0104                 ;
0105 FD84 3E1B       EXIT:  LD    A,27           ;LPRINT CHR$(27,37,57,0)
0106 FD86 CDE212            CALL  LPTOUT         ;ｸﾞﾗﾌｨｯｸ ｼﾃｲ ﾉ ｶｲｼﾞｮ
0107 FD89 3E25              LD    A,37
0108 FD8B CDE212            CALL  LPTOUT
0109 FD8E 3E39              LD    A,57
0110 FD90 CDE212            CALL  LPTOUT
0111 FD93 AF                XOR   A
0112 FD94 CDE212            CALL  LPTOUT
0113 FD97 C9                RET
0114                 ;
0115                 ;
0116 FD98 00000000          DB    00H,00H,00H,00H
0117 FD9C 00000000          DB    00H,00H,00H,00H
0118                 ;
0119                 ;
0120                 ;
0121                 ;**************************
0122                 ;
0123                 ;   ﾃｷｽﾄ ｶﾞﾒﾝ ﾊｰﾄﾞ ｺﾋﾟｰ
0124                 ;
0125                 ;       by Y.Shimizu
0126                 ;
0127                 ;       1984.6.10 SUN
0128                 ;
0129                 ;**************************
0130                 ;
0131                 ;
0132                        ORG   0FDA0H
0133                 ;
0134                 ;
0135                 ;------------------------------------------
0136                 ;     text bit pattern setting subroutine
0137                 ;------------------------------------------
0138                 ;
0139 FDA0 CD30FE     PATST1:CALL  ADCAL
```

```
0140 FDA3 010020          LD    BC,2000H        ;ｱﾄﾘﾋﾞｭｰﾄ VRAM ｾﾝﾄｳ ｱﾄﾞﾚｽ
0141 FDA6 09              ADD   HL,BC
0142 FDA7 E5              PUSH  HL
0143 FDA8 C1              POP   BC              ;BC=ｱﾄﾘﾋﾞｭｰﾄ VRAM ｱﾄﾞﾚｽ
0144                 ;
0145 FDA9 ED58       PEEK:   IN    E,(C)         ;E =ｱﾄﾘﾋﾞｭｰﾄ ｺｰﾄﾞ
0146 FDAB CBE0            SET   4,B             ;BC=ﾃｷｽﾄ VRAM ｱﾄﾞﾚｽ
0147 FDAD ED50            IN    D,(C)           ;D =ｷｬﾗｸﾀ ｺｰﾄﾞ
0148                 ;
0149 FDAF 2188FE     CGCHK: LD    HL,BUFFER      ;HL=ﾊﾞｯﾌｧ ｾﾝﾄｳ ｱﾄﾞﾚｽ
0150 FDB2 D5              PUSH  DE              ;VRAM ｺｰﾄﾞ ｦ PUSH
0151 FDB3 CB6B            BIT   5,E             ;ｱﾄﾘﾋﾞｭｰﾄ CG ﾁｪｯｸ
0152 FDB5 2825            JR    Z,CGEN0
0153                 ;
0154 FDB7 1E15       CGEN1: LD    E,15H          ;read RAMCG(B)
0155 FDB9 CD3300          CALL  PCGRD
0156 FDBC 1E16            LD    E,16H           ;read RAMCG(R)
0157 FDBE CD3300          CALL  PCGRD
0158 FDC1 1E17            LD    E,17H           ;read RAMCG(G)
0159 FDC3 CD3300          CALL  PCGRD
0160                 ;
0161                 ;
0162 FDC6 0E02       BITOR: LD    C,2            ;ﾙｰﾌﾟ ｶｳﾝﾀ
0163                 ;
0164 FDC8 11F8FF     LPOR:  LD    DE,-8
0165 FDCB EB              EX    DE,HL
0166 FDCC 19              ADD   HL,DE           ;DE=HL,HL=HL-8
0167 FDCD 0608            LD    B,8             ;ﾙｰﾌﾟ ｶｳﾝﾀ
0168                 ;
0169 FDCF 1B         LPOR1: DEC   DE
0170 FDD0 2B              DEC   HL
0171 FDD1 1A              LD    A,(DE)
0172 FDD2 B6              OR    (HL)
0173 FDD3 77              LD    (HL),A          ;(HL)=(HL) OR (DE)
0174 FDD4 10F9            DJNZ  LPOR1
0175                 ;
0176 FDD6 EB              EX    DE,HL
0177 FDD7 0D              DEC   C
0178 FDD8 20EE            JR    NZ,LPOR
0179                 ;
0180 FDDA 1805            JR    CREVCK
0181                 ;
0182 FDDC 1E14       CGEN0: LD    E,14H          ;read ROMCG
0183 FDDE CD3300          CALL  PCGRD
0184                 ;
0185 FDE1 D1         CREVCK:POP   DE             ;DE=VRAM ｺｰﾄﾞ POP
0186 FDE2 CB5B            BIT   3,E             ;ｱﾄﾘﾋﾞｭｰﾄ CREV ﾉ ﾁｪｯｸ
0187 FDE4 280B            JR    Z,EXTSB
0188                 ;
0189 FDE6 2188FE     CREV1: LD    HL,BUFFER      ;CREV1 ﾉ ｼｮﾘ
0190 FDE9 0608            LD    B,8             ;ﾙｰﾌﾟ ｶｳﾝﾀ
0191                 ;
0192 FDEB 7E         LPREV: LD    A,(HL)
0193 FDEC 2F              CPL                   ;ﾋﾞｯﾄ ﾉ ﾊﾝﾃﾝ
0194 FDED 77              LD    (HL),A
0195 FDEE 23              INC   HL
0196 FDEF 10FA            DJNZ  LPREV
0197                 ;
0198 FDF1 CD60FE     EXTSB: CALL  TRANS          ;ﾋﾞｯﾄ ｦ ﾃﾝﾁ ｽﾙ
0199                 ;
0200 FDF4 C9              RET
0201                 ;
0202                 ;
0203 FDF5 00000000        DB    00H,00H,00H,00H
0204 FDF9 00000000        DB    00H,00H,00H,00H
0205 FDFD 000000          DB    00H,00H,00H
0206                 ;
0207                 ;
0208                 ;***************************
0209                 ;
0210                 ;   ｸﾞﾗﾌｨｯｸ ｶﾞﾒﾝ ﾊｰﾄﾞ ｺﾋﾟｰ
0211                 ;
0212                 ;        by  Y.Shimizu
0213                 ;
0214                 ;        1984.6.12 TUE
0215                 ;
0216                 ;***************************
0217                 ;
0218                 ;
0219                      ORG   0FE00H
0220                 ;
0221                 ;
0222 FE00 010040     PATST2:LD    BC,4000H       ;GRAM blue  ｾﾝﾄｳ ｱﾄﾞﾚｽ
0223 FE03 CD13FE          CALL  GRAPAT
0224 FE06 010080          LD    BC,8000H        ;GRAM red   ｾﾝﾄｳ ｱﾄﾞﾚｽ
0225 FE09 CD13FE          CALL  GRAPAT
0226 FE0C 0100C0          LD    BC,0C000H       ;GRAM green ｾﾝﾄｳ ｱﾄﾞﾚｽ
0227 FE0F CD13FE          CALL  GRAPAT
```

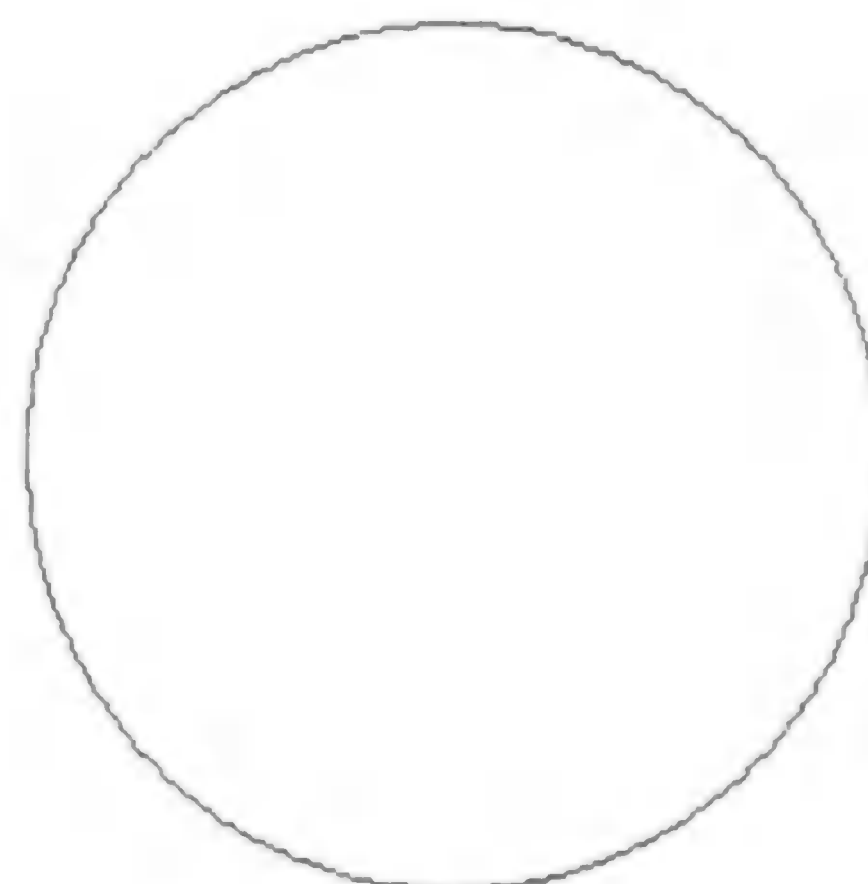
```
0228                    ;
0229 FE12 C9            ;       RET
0230                    ;
0231                    ;
0232                    ;
0233                    ;-------------------------------------------------
0234                    ;    graphic bit pattern setting subroutine
0235                    ;-------------------------------------------------
0236                    ;
0237                    ;    input condition ...  BC = GRAM top address
0238                    ;                                blue   ==> 4000H
0239                    ;                                red    ==> 8000H
0240                    ;                                green  ==> C000H
0241                    ;
0242                    ;    ...............................................
0243                    ;
0244                    ;
0245 FE13 CD30FE        GRAPAT:CALL ADCAL             ;アト゛レス ケイサン
0246 FE16 09                   ADD  HL,BC             ;HL= GRAM セ゛ッタイ アト゛レス
0247                    ;
0248 FE17 1188FE        GETGR: LD   DE,BUFFER         ;DE= ハ゛ッファ セントウ アト゛レス
0249 FE1A 0608                 LD   B,8               ;ルーフ゜ カウンタ
0250                    ;
0251 FE1C C5            GETLP: PUSH BC                ;カウンタ ヲ PUSH
0252 FE1D 4D                   LD   C,L
0253 FE1E 44                   LD   B,H               ;BC= GRAM アト゛レス
0254 FE1F ED78                 IN   A,(C)             ;PEEK@
0255 FE21 12                   LD   (DE),A            ;ハ゛ッファ ニ ストア
0256 FE22 13                   INC  DE                ;ハ゛ッファ アト゛レス ヲ +1 スル
0257 FE23 210008               LD   HL,0800H          ;ラスタ カン ノ GRAM アト゛レス ステッフ゜
0258 FE26 09                   ADD  HL,BC             ;HL =ツキ゛ ノ ラスタ ノ GRAM アト゛レス
0259 FE27 C1                   POP  BC                ;カウンタ ヲ POP
0260 FE28 10F2                 DJNZ GETLP
0261                    ;
0262 FE2A CD60FE               CALL TRANS             ;ヒ゛ット ヲ テンチ
0263                    ;
0264 FE2D C9                   RET
0265                    ;
0266                    ;
0267 FE2E 0000                 DB   00H,00H
0268                    ;
0269                    ;
0270                    ;**********************
0271                    ;
0272                    ;   キョウツウ リヨウ サフ゛ルーチン
0273                    ;
0274                    ;**********************
0275                    ;
0276                    ;
0277                           ORG  0FE30H
0278                    ;
0279                    ;
0280                    ;------------------------------
0281                    ;    VRAM アト゛レス ケイサン サフ゛ルーチン
0282                    ;------------------------------
0283                    ;
0284                    ;    output condition ... HL= VRAM ソウタイ アト゛レス
0285                    ;
0286                    ;    function ...  (XLOC),(YLOC) カラ VRAM ソウタイ アト゛レス
0287                    ;                  ヲ ケイサン スル。
0288                    ;
0289                    ;    ..............................................
0290                    ;
0291                    ;
0292 FE30 DD21A0FE      ADCAL: LD   IX,XLOC           ;VRAM アト゛レス ケイサン
0293 FE34 DD6E01               LD   L,(IX+1)          ;HL=Y
0294 FE37 2600                 LD   H,0
0295 FE39 DD5E00               LD   E,(IX+0)          ;DE=X
0296 FE3C 1600                 LD   D,0
0297                    ;
0298 FE3E 29                   ADD  HL,HL
0299 FE3F 29                   ADD  HL,HL
0300 FE40 29                   ADD  HL,HL             ;Y ヲ 8 ハ゛イ スル
0301                    ;
0302 FE41 3A0700        WIDCHK:LD   A,(WIDTH)
0303 FE44 FE28                 CP   40
0304 FE46 2806                 JR   Z,WID40
0305                    ;
0306 FE48 29            WID80: ADD  HL,HL             ;Y ハ 16 ハ゛イ
0307 FE49 E5                   PUSH HL                ;16*Y ヲ PUSH
0308 FE4A 29                   ADD  HL,HL
0309 FE4B 29                   ADD  HL,HL             ;Y ハ 64 ハ゛イ
0310 FE4C 180D                 JR   COMMON
0311                    ;
0312 FE4E E5            WID40: PUSH HL                ;8*Y ヲ PUSH
0313 FE4F 29                   ADD  HL,HL
0314 FE50 29                   ADD  HL,HL             ;Y ハ 32 ハ゛イ
0315 FE51 D5                   PUSH DE                ;DE (X ザ゛ヒョウ) ヲ PUSH
```

```
0316 FE52 ED5BE900          LD    DE,(SCRPG)      ;DE= 0 or 4 (screen page)
0317 FE56 7B                LD    A,E
0318 FE57 5A                LD    E,D
0319 FE58 57                LD    D,A             ;DE= VRAM ﾊﾞｲｱｽ (SWAP D,E)
0320 FE59 19                ADD   HL,DE           ;HL= HL+ VRAM ﾊﾞｲｱｽ (0 or 400H)
0321 FE5A D1                POP   DE              ;DE= X ｻﾞﾋｮｳ
0322                 ;
0323 FE5B 19         COMMON:ADD   HL,DE           ;HL= HL+X
0324 FE5C D1                POP   DE              ;DE= 8*Y or 16*Y
0325 FE5D 19                ADD   HL,DE           ;HL= VRAM ｿｳﾀｲ ｱﾄﾞﾚｽ
0326                 ;
0327 FE5E C9                RET
0328                 ;
0329                 ;
0330 FE5F 00                DB    00H
0331                 ;
0332                 ;
0333                 ;----------------------------------
0334                 ;    ﾋﾞｯﾄ ﾊﾟﾀｰﾝ ｦ ﾃﾝﾁ ｽﾙ ｻﾌﾞﾙｰﾁﾝ
0335                 ;----------------------------------
0336                 ;
0337 FE60 2188FE     TRANS: LD    HL,BUFFER       ;ﾋﾞｯﾄ ﾉ ﾃﾝﾁ ｦ ｽﾙ ﾙｰﾁﾝ
0338 FE63 1180FE            LD    DE,BITPAT
0339 FE66 0E08              LD    C,8             ;ﾙｰﾌﾟ ｶｳﾝﾀ
0340                 ;
0341 FE68 0608       LPTR:  LD    B,8             ;ﾙｰﾌﾟ ｶｳﾝﾀ
0342 FE6A E5                PUSH  HL
0343 FE6B AF                XOR   A
0344                 ;
0345 FE6C CB16       LPTR1: RL    (HL)
0346 FE6E 17                RLA                   ;A ﾚｼﾞ ﾆ ｶｸ MSB ｦ ｱﾂﾒﾙ
0347 FE6F 23                INC   HL
0348 FE70 10FA              DJNZ  LPTR1
0349                 ;
0350 FE72 E1                POP   HL
0351 FE73 EB                EX    DE,HL
0352 FE74 B6                OR    (HL)
0353 FE75 77                LD    (HL),A          ;BUFFER ﾉ ﾃﾝﾁ ｦ BITPAT ﾆ OR ｽﾙ
0354 FE76 EB                EX    DE,HL
0355 FE77 13                INC   DE
0356 FE78 0D                DEC   C
0357 FE79 20ED              JR    NZ,LPTR
0358                 ;
0359 FE7B C9                RET
0360                 ;
0361                 ;
0362                 ;
0363                 ;-----------------------------------
0364                 ;    WORKING AREA of this routine
0365                 ;-----------------------------------
0366                 ;
0367                 ;
0368                        ORG   0FE80H
0369                 ;
0370                 ;
0371 FE80            BITPAT:DS    8               ;ﾋﾞｯﾄ ﾊﾟﾀｰﾝ ｶｸﾉｳ ｴﾘｱ ( 8 ﾊﾞｲﾄ )
0372 FE88            BUFFER:DS    24              ;ｻｷﾞｮｳ ﾖｳ ﾊﾞｯﾌｧ  ( 24 ﾊﾞｲﾄ )
0373 FEA0            XLOC:  DS    1               ;ｶﾞﾒﾝ X ｻﾞﾋｮｳ  ( 1 ﾊﾞｲﾄ )
0374 FEA1            YLOC:  DS    1               ;ｶﾞﾒﾝ Y ｻﾞﾋｮｳ  ( 1 ﾊﾞｲﾄ )
0375                 ;
0376                 ;
0377 FEA2                   END
```

チェックサムリスト

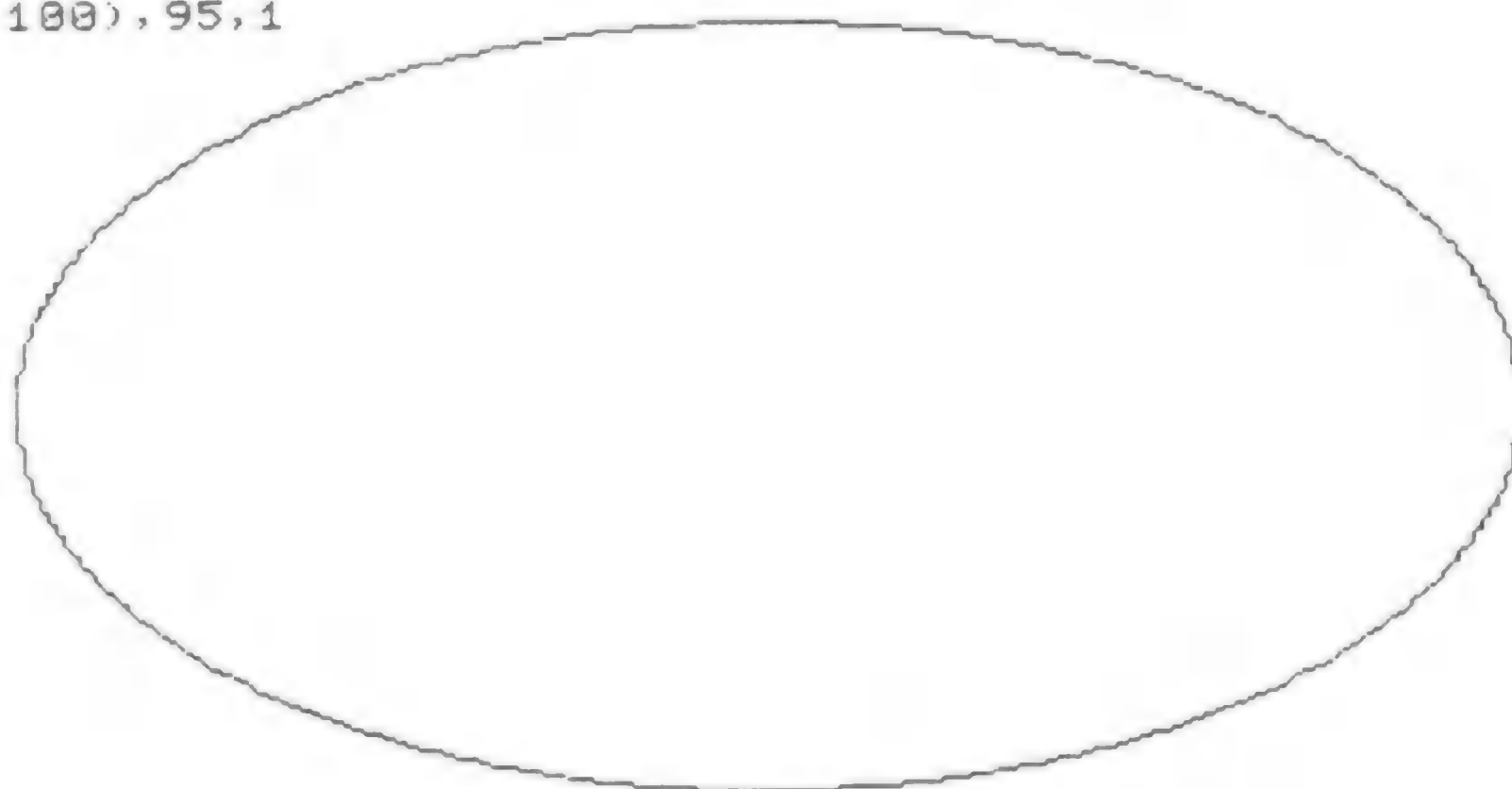
```
:FD00 = 21 A1 FE 36 00 3E 1B CD  : 1C
:FD08 = E2 12 3E 25 CD E2 12 3E  : 56
:FD10 = 39 CD E2 12 3E 0F CD E2  : F6
:FD18 = 12 21 A0 FE 36 00 3A 36  : 77
:FD20 = 00 FE 03 CA 84 FD 3A 07  : 8D
:FD28 = 00 FE 50 28 05 21 40 01  : DD
:FD30 = 18 03 21 80 02 3E 1B CD  : E4
:FD38 = E2 12 3E 25 CD E2 12 3E  : 56
:FD40 = 32 CD E2 12 7C CD E2 12  : 30
:FD48 = 7D CD E2 12 21 80 FE AF  : 8C
:FD50 = 06 08 77 23 10 FC CD A0  : 21
:FD58 = FD 00 00 00 21 80 FE 06  : A2
:FD60 = 08 7E CD E2 12 23 10 F9  : 73
:FD68 = 3A 07 00 47 21 A0 FE 34  : 7B
:FD70 = 7E B8 DA 4C FD 3E 0A CD  : 6E
:FD78 = E2 12 21 A1 FE 34 7E FE  : 64
-------------------------------------
        9C A3 73 5F 95 6B 1C 95  : C2

:FD80 = 19 DA 19 FD 3E 1B CD E2  : 11
:FD88 = 12 3E 25 CD E2 12 3E 39  : AD
:FD90 = CD E2 12 AF CD E2 12 C9  : FA
:FD98 = 00 00 00 00 00 00 00 00  : 00
:FDA0 = CD 30 FE 01 00 20 09 E5  : 0A
:FDA8 = C1 ED 58 CB E0 ED 50 21  : 0F
:FDB0 = 88 FE D5 CB 6B 28 25 1E  : FC
:FDB8 = 15 CD 33 00 1E 16 CD 33  : 49
:FDC0 = 00 1E 17 CD 33 00 0E 02  : 45
:FDC8 = 11 F8 FF EB 19 06 08 1B  : 35
:FDD0 = 2B 1A B6 77 10 F9 EB 0D  : 73
:FDD8 = 20 EE 18 05 1E 14 CD 33  : 5D
:FDE0 = 00 D1 CB 5B 28 0B 21 88  : D3
:FDE8 = FE 06 08 7E 2F 77 23 10  : 63
:FDF0 = FA CD 60 FE C9 00 00 00  : EE
:FDF8 = 00 00 00 00 00 00 00 00  : 00
-------------------------------------
        77 A4 C5 1B F0 EF 7A 30  : 84

:FE00 = 01 00 40 CD 13 FE 01 00  : 20
:FE08 = 80 CD 13 FE 01 00 C0 CD  : EC
:FE10 = 13 FE C9 CD 30 FE 09 11  : EF
:FE18 = 88 FE 06 08 C5 4D 44 ED  : D7
:FE20 = 78 12 13 21 00 08 09 C1  : 90
:FE28 = 10 F2 CD 60 FE C9 00 59  : 4F
:FE30 = DD 21 A0 FE DD 6E 01 26  : 0E
:FE38 = 00 DD 5E 00 16 00 29 29  : A3
:FE40 = 29 3A 07 00 FE 28 28 06  : BE
:FE48 = 29 E5 29 29 18 0D E5 29  : 93
:FE50 = 29 D5 ED 5B E9 00 7B 5A  : 04
:FE58 = 57 19 D1 19 D1 19 C9 00  : 0D
:FE60 = 21 88 FE 11 80 FE 0E 08  : 4C
:FE68 = 06 08 E5 AF CB 16 17 23  : BD
:FE70 = 10 FA E1 EB B6 77 EB 13  : 01
:FE78 = 0D 20 ED C9 00 00 00 00  : E3
-------------------------------------
        97 82 9F 30 CB 61 A2 FB  : B1
```

WIDTH 40でのハード・コピー

```
SHARP-HuBASIC CZ-8FB01 V1.0
Copyright (C) 1982 by SHARP/Hudson

20989 Bytes free

Ok
CLEAR &HFD00
Ok
MON
*L
Found  "HCOPY 0,4     .    "
*GFD00
```

```
INIT:CIRCLE(160,100),95,1
Ok
MON
*MFD56
:FD56=CD
:FD57=A0
:FD58=FD
:FD59=CD
:FD5A=00
:FD5B=FE
:FD5C=21
*GFD00
```

WIDTH 80 でのハード・コピー

```
*M11A2
:11A2=10
:11A3=CD
*M119D
:119D=FF
:119E=00
*DFD00
:FD00=21  A1  FE  36  00  3E  1B  CD  E2  12  3E  25  CD  E2  12  3E  /!.生6.>.へ♠.>%へ♠.>
:FD10=39  CD  E2  12  3E  0F  CD  E2  12  21  A0  FE  36  00  3A  36  /9へ♠.>.へ♠.! 生6.:6
:FD20=00  FE  03  CA  84  FD  3A  07  00  FE  50  28  05  21  40  01  /.生.ハ■人:..生P(.!@.
:FD30=18  03  21  80  02  3E  1B  CD  E2  12  3E  25  CD  E2  12  3E  /..!_.>.へ♠.>%へ♠.>
:FD40=32  CD  E2  12  7C  CD  E2  12  7D  CD  E2  12  21  80  FE  AF  /2へ♠.|へ♠.}へ♠.!_生ッ
:FD50=06  00  77  23  10  FC  CD  A0  FD  CD  00  FE  21  80  FE  06  /..w#.円へ 人へ.生!_生.
:FD60=08  7E  CD  E2  12  23  10  F9  3A  07  00  47  21  A0  FE  34  /.~へ♠.#.分:..G! 生4
:FD70=7E  88  DA  4C  FD  3E  0A  CD  E2  12  21  A1  FE  34  7E  FE  /~クレL人>.へ♠.!.生4~生
:FD80=19  DA  19  FD  3E  1B  CD  E2  12  3E  25  CD  E2  12  3E  39  /.レ.人>.へ♠.>%へ♠.>9
:FD90=CD  E2  12  AF  CD  E2  12  C9  00  00  00  00  00  00  00  00  /へ♠.ッへ♠.J........
:FDA0=CD  30  FE  01  00  20  09  E5  C1  ED  58  CB  E0  ED  50  21  /へ0生.. .♣チ■XE■■P!
:FDB0=88  FE  D5  CB  6B  28  25  1E  15  CD  33  00  1E  16  CD  33  /| 生コEk(%..へ3...へ3
:FDC0=00  1E  17  CD  33  00  0E  02  11  F8  FF  EB  19  06  08  1B  /...へ3.....畋〒■.....
:FDD0=28  1A  B6  77  10  F9  EB  0D  20  EE  18  05  1E  14  CD  33  /+.カw.分■. ■.....へ3
:FDE0=00  D1  CB  5B  28  0B  21  88  FE  06  08  7E  2F  77  23  10  /.ムE[(.!| 生..~/w#.
:FDF0=FA  CD  60  FE  C9  00  00  00  00  00  00  00  00  00  00  00  /秒へ`生J...........
*GFD00
```

```
INIT:CIRCLE(320,100),95,1
Ok
CALL&HFD00
```

第6章　サブCPU

6－1　80C49と80C48

X 1には、80C49及び80C48という2個のLSIが、サブCPUとして搭載されています。これらのLSIはいずれも、マイクロコンピュータに必要な機能であるCPU, ROM, RAM, 入出力ポート、タイマ等を1個のチップに集積した、いわゆる「ワンチップ・マイコン」の仲間です。80C49と80C48とは同型のLSIであって、これらは内部に有するメモリー容量により区別されます。X 1においては、80C49は本体内に、80C48はキーボード内に配置されています。

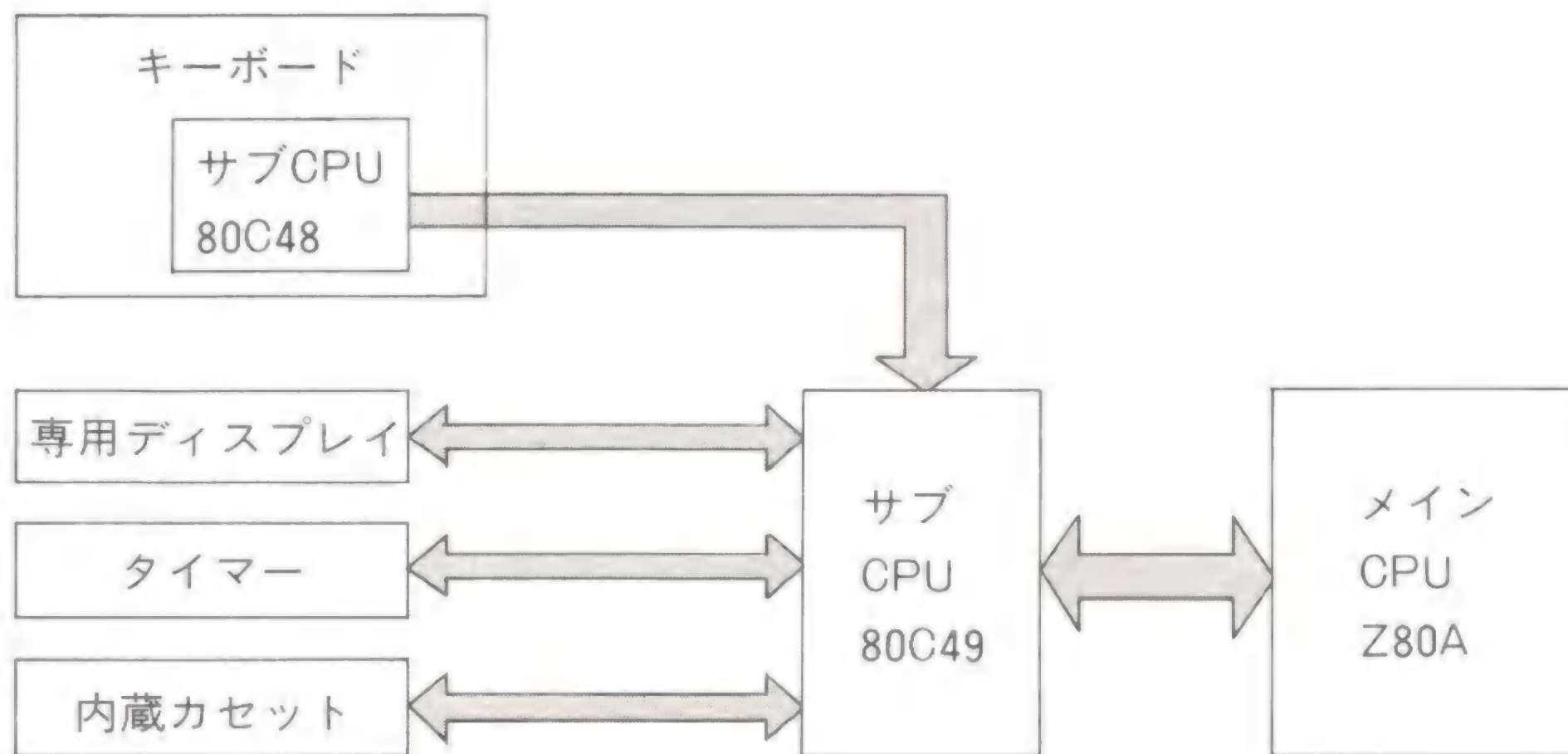
ワンチップ・マイコン　　80C49, 80C48

	80C49	80C48
内部ROM	2Kバイト	1Kバイト
内部RAM	128バイト	64バイト
X1での役割	本体内に搭載。 ①キーボードからの直列信号を受信し、並列データに直す。 ②専用ディスプレイテレビ、タイマー、内蔵カセットを制御する。	キーボード内に搭載。 押されたキーのデータを直列信号にしてX1本体へ送信する。

〔注〕80C49、80C48の真中のCは、このLSIがC-MOSとよばれる技術でできていることを示します。C-MOSのLSIは、消費電力が小さく、X 1のサブCPUは本体のパワー・スイッチOFFの時でも、背面のメイン・スイッチさえ入れておけば、わずかな電力で動いています。キーボードからテレビを制御できるのはそのためです。

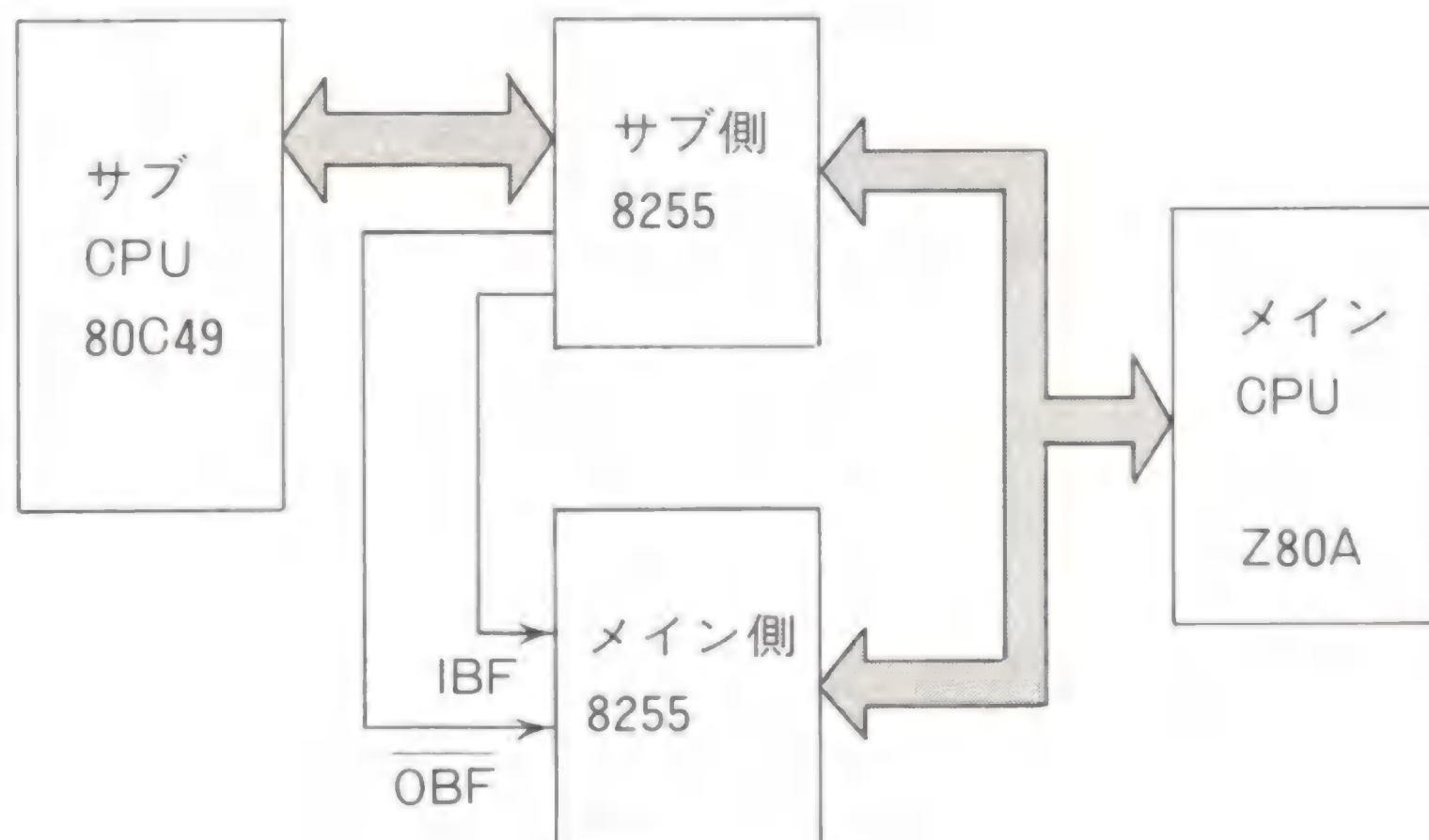
6－2　サブCPUの位置づけ

X1の2個のサブCPUと、メインCPU、周辺装置との関係はおおよそ次のようになっています。

キーボード
サブCPU
80C48
専用ディスプレイ
タイマー
内蔵カセット
サブ
CPU
80C49
メイン
CPU
Z80A

これらのサブCPUは、内蔵ROM内のプログラムに従って、メインCPUとは非同期で、動作をしています。従って、サブCPUが管理しているキーボード、専用ディスプレイ、タイマー、内蔵カセットとのデータ授受をするためには、一定の方法で、サブCPU 80C49 との交信を行なう必要があります。

6－3　サブCPUとの交信法

メインCPU (Z80A) と、サブCPU 80C49 とは、2個のインターフェースLSI 8255を経由して、互いに交信します。これらの 8255 については、第4章で詳しく説明しましたから参照して下さい。

サブ
CPU
80C49
サブ側
8255
メイン
CPU
Z80A
IBF
メイン側
8255
OBF

これらに関連する入出力ポートと、そのI／Oアドレスは以下の通りです。

サブCPU関係のI／Oアドレス

関連ポート	I／Oアドレス	意味	
サブ側8255 ポートA	1900H （注）	サブCPU80C49へのデータ入出力ポート。	
メイン側 8255 ポートB	1A01H （注）	ビット5	サブ側8255ポートAの$\overline{\text{OBF}}$（Output Buffer Full）信号が来ている。この信号が“L”のとき、サブCPUからのデータがサブ側8255ポートAにたまったことを意味する。
		ビット6	サブ側8255ポートAのIBF（Input Buffer Full）信号が来ている。この信号が“L”のとき、サブCPUへデータを出力することができる。

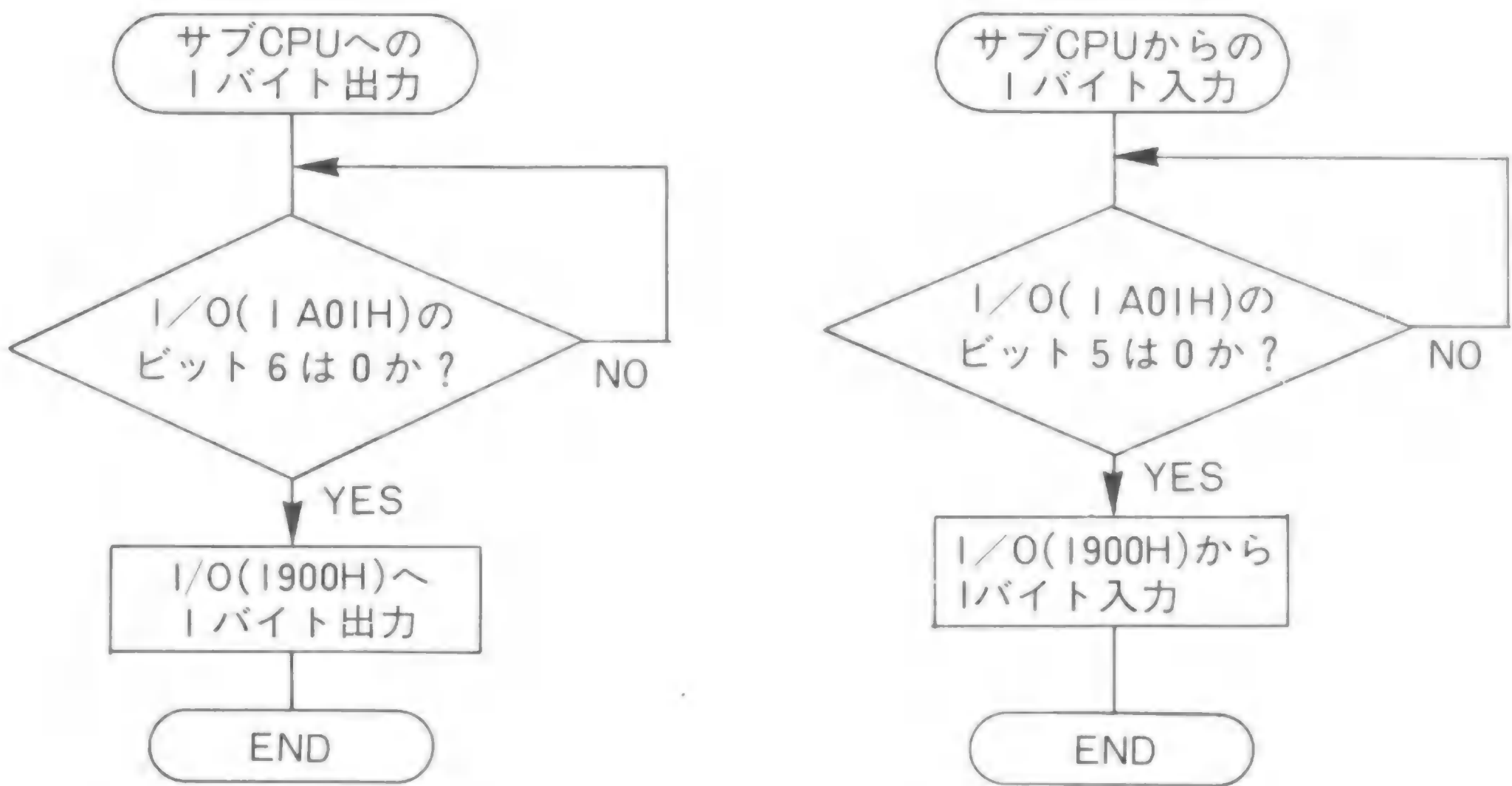
（注）これらのアドレスは、あくまで一例です。4の倍数の違いは無視されるので、たとえば、1A05H番地でも、1A01H番地と同様になります。

サブCPU 80C49が、サブ側8255にデータを出力すると、$\overline{\text{OBF}}$信号が“L”となり、データはポートAに出てきます。$\overline{\text{OBF}}$信号は、メイン側8255ポートBのビット5に来ているので、このビットが0になるのを確かめてから、データを取り込めば、サブCPUからのデータが得られます。

サブCPU 80C49は、次のデータ受信準備完了になると、IBF信号を“L”にします。データ処理中は、IBFが“H”なので、メイン側からサブ側へデータを出力することはできません。このIBF信号は、メイン側8255ポートBのビット6に来ているので、このビットが0になるのを確認してから、サブCPUへデータを出力する必要があります。

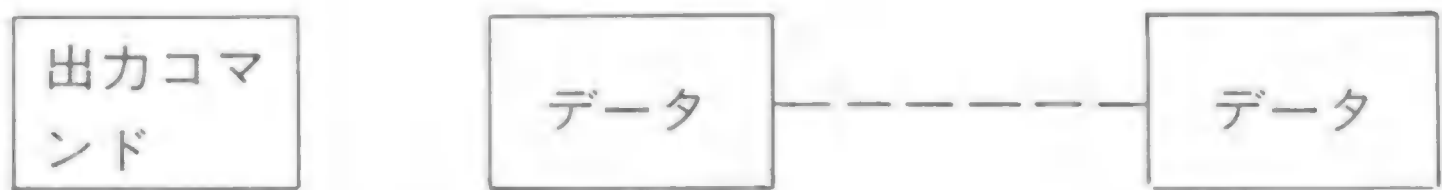
従って、サブCPUとの1バイト・データのやりとりの手続きを流れ図にすると以下のようになります。

サブCPUへの1バイト出力
I／O(1A01H)のビット6は0か？
NO
YES
I／O(1900H)へ1バイト出力
END

サブCPUからの1バイト入力
I／O(1A01H)のビット5は0か？
NO
YES
I／O(1900H)から1バイト入力
END

6－4　サブCPU出力コマンド

サブCPU 80C49を介して、専用ディスプレイテレビ、タイマー、内蔵カセットをコントロールするには、次のようなデータ列をサブCPUに出力する必要があります。

《サブCPUへのデータ出力》

出力コマンド	データ	------	データ

データは、1、3、6バイトのいずれか。データのバイト数と書式は、出力コマンドにより定まっている。

サブCPUへのデータ出力用コマンドと、それに続くデータ列の書式を以下にまとめました。

サブCPUへのデータ出力用コマンド一覧

コマンド(16進)	続くデータのバイト数	意味とデータ書式
D1	6	専用ディスプレイテレビ用 TVタイマー1のデータ設定 1バイト目(セットorリセット)……タイマーセットは40H、リセットは00H 2バイト目(ONorOFF)……OFFは0DH、ONチャンネルnは、8FH+n リセットは00H 3バイト目(分) 4バイト目(時)} 2進化10進数(BCD)により値を設定。(注1) 5バイト目(月と曜)……上位4ビットが月、下位4ビットが曜。(注2) 6バイト目(日)→2進化10進数により値を設定。(注1)
D2	6	専用ディスプレイテレビ用 TV タイマー2のデータ設定 (タイマー1に同じ)
D3	6	専用ディスプレイテレビ用 TV タイマー3のデータ設定 (タイマー1に同じ)
D4	6	専用ディスプレイテレビ用 TV タイマー4のデータ設定 (タイマー1に同じ)

<table>
<tr><td>D 5</td><td>6</td><td>専用ディスプレイテレビ用 TV タイマー 5 のデータ設定
（タイマー 1 に同じ）</td></tr>
<tr><td>D 6</td><td>6</td><td>専用ディスプレイテレビ用 TV タイマー 6 のデータ設定
（タイマー 1 に同じ）</td></tr>
<tr><td>D 7</td><td>6</td><td>専用ディスプレイテレビ用 TV タイマー 7 のデータ設定
（タイマー 1 に同じ）</td></tr>
<tr><td>E 4</td><td>1</td><td>サブCPUの割り込みベクトルの設定
サブCPUがメインCPUに、モード 2 の割り込みをかけるときの、ジャンプテーブル格納エリア下位アドレス 1 バイトをセットする。キー入力割り込み用である。</td></tr>
<tr><td>E 7</td><td>1</td><td>専用ディスプレイテレビのコントロール
データ　01H　＝音量アップ
02H　＝音量ダウン
03H　＝標準音量
06H　＝音声ミュートのON／OFF反転
09H　＝チャンネル・コールのON／OFF反転
0BH　＝チャンネル順送り
0CH　＝チャンネル逆送り
0DH、0EH　＝パワーOFF
10H　＝チャンネル 1
11H　＝チャンネル 2
⋮　⋮
1AH　＝チャンネル11
1BH　＝チャンネル12
（10H〜1BH：0FH ＋チャンネル番号）
1CH　＝テレビ画面にする
1DH　＝コンピュータ画面にする
1EH　＝ テレビ画面のコントラストを下げて、スーパーインポーズ
1FH　＝ そのままスーパーインポーズ
以上のデータに80Hを加えたものは、パワーON後、指定された動作をします。たとえば、9BHは、12チャンネルでONします。単に80Hは、パワー ON のみをします。</td></tr>
<tr><td>E 9</td><td>1</td><td>内蔵カセットのコントロール
データ00H＝EJECT
01H＝STOP
02H＝PLAY（READ）
03H＝FAST
04H＝REW
05H＝APSS 1
06H＝APSS−1
0AH＝RECORD（WRITE）</td></tr>
</table>

EC	3	タイマーに年月日を設定 1バイト目（年）………2進化10進数により値を設定。（注1） 2バイト目（月と曜）…上位4ビットが月、下位4ビットが曜（注2） 3バイト目（日）………2進化10進数により値を設定。（注1）
EE	3	タイマーに時分秒を設定 1バイト目（時） 2バイト目（分） 2進化10進数により値を設定。 3バイト目（秒）（注1）

（注1）たとえば10進数12をセットするには、そのまま16進数と見なして、12Hをセットするのが、2進化10進数（BCD）による表現です。

（注2）上位4ビットの月のデータは16進表示されます。1月から9月は各々1～9、10月～12月は各々A～Cで表わします。下位4ビットの曜のデータは以下の通り。

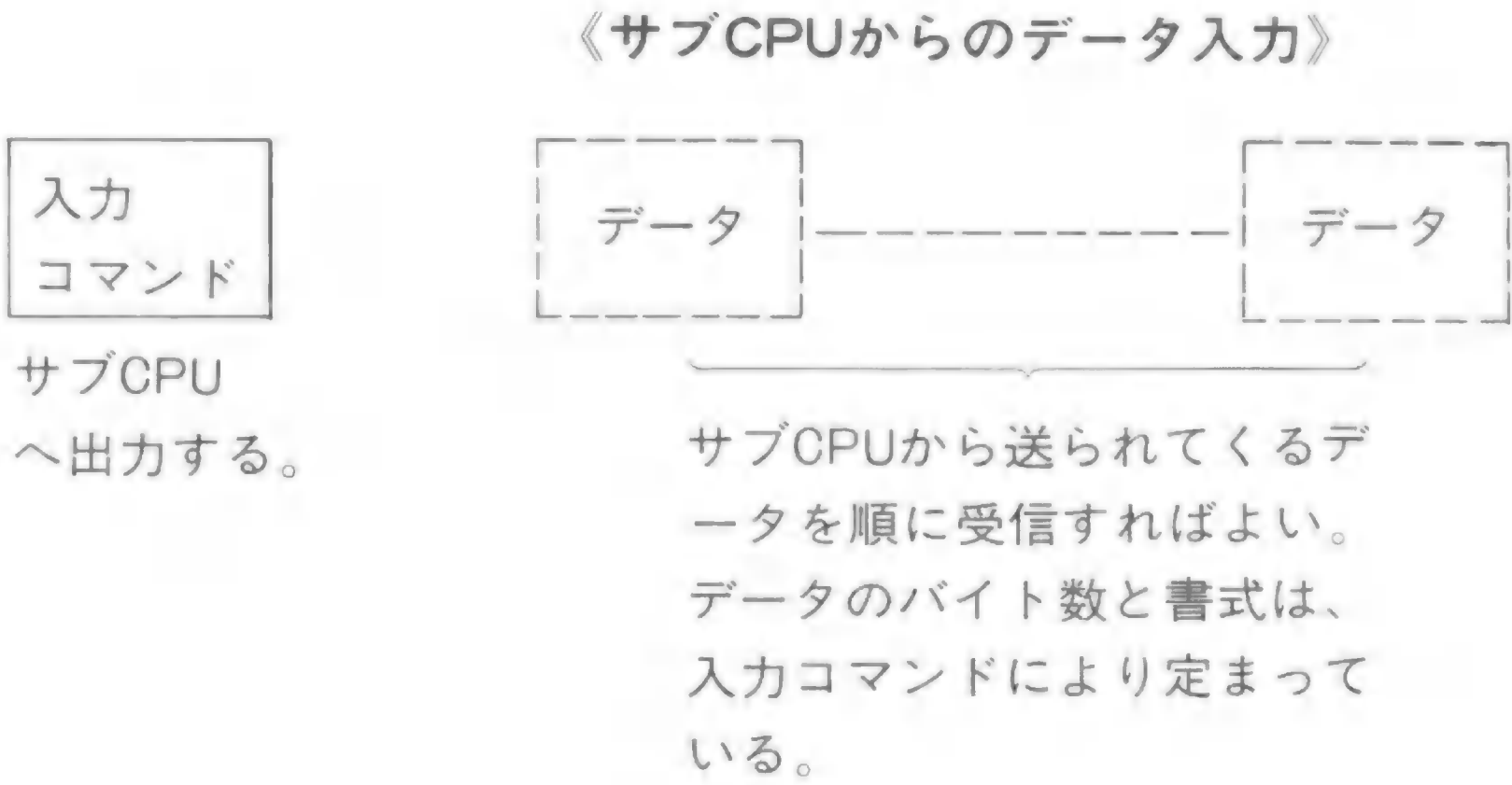
曜	SUN	MON	TUE	WED	THU	FRI	SAT
コード	0	1	2	3	4	5	6

たとえば、A3Hは10月と水曜日を意味するデータとなります。

6－5　サブCPU入力コマンド

今度は、サブCPUからのデータを受けとるコマンドについて説明します。出力コマンドと同様に、入力コマンドは1バイトからなり、コマンド毎にサブCPUから送られるデータのバイト数が定まっています。

《サブCPUからのデータ入力》

入力コマンド	データ	――――――	データ
サブCPUへ出力する。	サブCPUから送られてくるデータを順に受信すればよい。データのバイト数と書式は、入力コマンドにより定まっている。		

入力コマンドと、サブCPUから送られてくるデータ列の書式を以下に掲げます。

サブCPUからのデータ入力用コマンド一覧

<table>
<tr><th>コマンド(16進)</th><th>送られてくる
データ・バイト数</th><th>データの意味</th></tr>
<tr><td>D 9</td><td>6</td><td>専用ディスプレイテレビ用 TV タイマー 1 の設定内容
(6バイトの内容は出力コマンドと同様)</td></tr>
<tr><td>DA</td><td>6</td><td>専用ディスプレイテレビ用 TV タイマー 2 の設定内容</td></tr>
<tr><td>DB</td><td>6</td><td>専用ディスプレイテレビ用 TV タイマー 3 の設定内容</td></tr>
<tr><td>DC</td><td>6</td><td>専用ディスプレイテレビ用 TV タイマー 4 の設定内容</td></tr>
<tr><td>DD</td><td>6</td><td>専用ディスプレイテレビ用 TV タイマー 5 の設定内容</td></tr>
<tr><td>DE</td><td>6</td><td>専用ディスプレイテレビ用 TV タイマー 6 の設定内容</td></tr>
<tr><td>DF</td><td>6</td><td>専用ディスプレイテレビ用 TV タイマー 7 の設定内容</td></tr>
<tr><td>E 6</td><td>2</td><td>押されたキーの情報(80C48からの直列データを2バイトの並列データにしたもの)
1バイト目…押されたキーのファンクション部(INKEY$(2)相当)
ビット7 … ビット0
<table>
<tr><td>テンキー</td><td>キー入力</td><td>リピート</td><td>GRAPH</td><td>CAP LOCK</td><td>カナ</td><td>SHIFT</td><td>CTRL</td></tr>
<tr><td>有 0
無 1</td><td>有 0
無 1</td><td>ON 0
OFF 1</td><td>有 0
無 1</td><td>有 0
無 1</td><td>有 0
無 1</td><td>有 0
無 1</td><td>有 0
無 1</td></tr>
</table>
2バイト目…押されたキーのデータ部(ASCIIコード)</td></tr>
<tr><td>E 8</td><td>1</td><td>専用ディスプレイテレビ コントロール・コード設定値
(データの意味は出力コマンドと同じ)</td></tr>
<tr><td>EA</td><td>1</td><td>内蔵カセット コントロール・コード設定値(CMT相当)
(データの意味は出力コマンドと同じ)</td></tr>
<tr><td>EB</td><td>1</td><td>サブ側8255ポートBのデータを得る(8255の章を参照)。
<table>
<tr><td>ビット7</td><td>ビット6</td><td>ビット5</td><td>ビット4</td><td>ビット3</td><td>ビット2</td><td>ビット1</td><td>ビット0</td></tr>
<tr><td>OBF$_A$</td><td>ACK$_A$</td><td>READ DATA</td><td>1</td><td>0</td><td>消去防止ツメ</td><td>テープセット</td><td>テープ走行</td></tr>
<tr><td colspan="2">読み出す時は1にセット
サブ側8255ポートAのモニター用</td><td>↑カセットより</td><td>↑つねに1にセット</td><td>↑未使用</td><td>折ってある…0
折ってない…1</td><td>セットされている……1
セットされていない…0</td><td>走行中…1
停止中…0</td></tr>
</table>
(※ビット2〜0は、CMT(x)相当)</td></tr>
<tr><td>ED</td><td>3</td><td>タイマーの年・月・日 現在値
(3バイト・データ内容は出力コマンドと同様)</td></tr>
<tr><td>EF</td><td>3</td><td>タイマーの時・分・秒 現在値
(3バイトデータの内容は出力コマンドと同様)</td></tr>
</table>

6－6　サブCPUとのデータ入出力ルーチン

サブCPUとのデータのやりとりの仕組みを学習するために、以下サブルーチンの形にまとめてみましょう。

仕様を簡単に述べます。

名　称	サブCPUへのデータ出力ルーチン
アドレス	F000H
入力レジスタ	Aレジスタ　＝サブCPU出力コマンド Bレジスタ　＝出力データのバイト数 DEレジスタ＝出力データ格納開始アドレス
機　能	サブCPUへ出力コマンドを送った後、指定バイト数だけ、指定アドレスからのデータを出力する。

名　称	サブCPUからのデータ入力ルーチン
アドレス	F100H
入力レジスタ	Aレジスタ　＝サブCPU入力コマンド Bレジスタ　＝入力データのバイト数 DEレジスタ＝入力データ格納開始アドレス
機　能	サブCPUへ入力コマンドを送った後、指定バイト数だけデータを受けとり、指定アドレスから順にメモリーに格納する。

リスト　　サブCPU　I／O　サブルーチン

```
0000          ;******************************
0001          ;*                            *
0002          ;*   ｻﾌﾞ CPU I/O ｻﾌﾞﾙｰﾁﾝ      *
0003          ;*                            *
0004          ;*       by Y.Shimizu         *
0005          ;*        1984.5.30           *
0006          ;*                            *
0007          ;******************************
0008          ;
0009          ;
0010          ;=======================================
0011          ; OUTPUT to SUB-CPU
0012          ;         address     : F000H-
0013          ;         input reg.  : A  = sub command
0014          ;                       B  = bytes
0015          ;                       DE = data pointer
0016          ;
0017          ; INPUT from SUB-CPU
0018          ;         address     : F100H-
0019          ;         input reg.  : A  = sub command
0020          ;                       B  = bytes
0021          ;                       DE = data pointer
0022          ;=======================================
0023          ;
0024          ;
0025          ;
0026                ORG   0F000H
0027          ;
0028          ;---------------------
0029          ; DATA OUT to SUB-CPU
0030          ;---------------------
0031          ;
```

```
0032 F000 F5                PUSH AF             ;Acc (command) push
0033 F001 CD13F0            CALL OCHECK         ;wait until IBF is "L"
0034 F004 F1                POP  AF             ;Acc (command) pop
0035 F005 CD1FF0            CALL OUTSUB         ;command out to sub-CPU
0036                ;
0037 F008 CD13F0    OUTLP:  CALL OCHECK         ;wait until IBF is "L"
0038 F00B 1A                LD   A,(DE)         ;A = data for output
0039 F00C CD1FF0            CALL OUTSUB         ;data out to sub-CPU
0040 F00F 13                INC  DE             ;increment data pointer
0041 F010 10F6              DJNZ OUTLP          ;loop by B
0042                ;
0043 F012 C9                RET
0044                ;
0045                ;------------------------
0046                ; WAIT for OUTPUT to SUB
0047                ;------------------------
0048                ;
0049 F013 C5        OCHECK:PUSH BC              ;B (counter) push
0050 F014 01011A            LD   BC,1A01H       ;8255 port B
0051                ;
0052 F017 ED78      L1:     IN   A,(C)          ;A = I/O(1A01H)
0053 F019 CB77              BIT  6,A            ;bit 6 check (IBF)
0054 F01B 20FA              JR   NZ,L1          ;wait until bit 6 is "L"
0055                ;
0056 F01D C1                POP  BC             ;B (counter) pop
0057 F01E C9                RET
0058                ;
0059                ;-------------------
0060                ; OUTPUT to SUB-CPU
0061                ;-------------------
0062                ;
0063 F01F C5        OUTSUB:PUSH BC              ;B (counter) push
0064 F020 010019            LD   BC,1900H       ;port A of 8255 (sub side)
0065 F023 ED79              OUT  (C),A          ;output to sub-CPU from Acc
0066 F025 C1                POP  BC             ;B (counter) pop
0067 F026 C9                RET
0068                ;
0069                ;
0070                        ORG  0F100H
0071                ;
0072                ;----------------------
0073                ; DATA IN from SUB-CPU
0074                ;----------------------
0075                ;
0076 F100 F5                PUSH AF             ;Acc (command) push
0077 F101 CD13F0            CALL OCHECK         ;wait until IBF is "L"
0078 F104 F1                POP  AF             ;Acc (command) pop
0079 F105 CD1FF0            CALL OUTSUB         ;command out to sub-CPU
0080                ;
0081 F108 CD13F1    INLP:   CALL ICHECK         ;wait until OBF is "L"
0082 F10B CD1FF1            CALL INSUB          ;data input from sub-CPU to Acc
0083 F10E 12                LD   (DE),A         ;data store to memory
0084 F10F 13                INC  DE             ;increment data pointer
0085 F110 10F6              DJNZ INLP           ;loop by B
0086                ;
0087 F112 C9                RET
0088                ;
0089                ;--------------------------
0090                ; WAIT for INPUT from SUB
0091                ;--------------------------
0092                ;
0093 F113 C5        ICHECK:PUSH BC              ;B (counter) push
0094 F114 01011A            LD   BC,1A01H       ;8255 port B
0095                ;
0096 F117 ED78      L2:     IN   A,(C)          ;A = I/O(1A01H)
0097 F119 CB6F              BIT  5,A            ;bit 5 check (OBF)
0098 F11B 20FA              JR   NZ,L2          ;wait until bit 5 is "L"
0099                ;
0100 F11D C1                POP  BC             ;B (counter) pop
0101 F11E C9                RET
0102                ;
0103                ;--------------------
0104                ; INPUT from SUB-CPU
0105                ;--------------------
0106                ;
0107 F11F C5        INSUB:  PUSH BC             ;B (counter) push
0108 F120 010019            LD   BC,1900H       ;port A of 8255 (sub side)
0109 F123 ED78              IN   A,(C)          ;input to Acc from sub-CPU
0110 F125 C1                POP  BC             ;B (counter) pop
0111 F126 C9                RET
0112                ;
0113                ;
0114 F127                   END
```

このサブルーチンを利用して、3つの実験をしてみます。

〔実験1〕

5番タイマーをセットします。TVタイマーを直接セットする命令は、HuBASICのコマンドにはありません。ASK命令で会話型のTVタイマー設定ルーチンを呼び出してからでなければいけません。

〔実験2〕

専用ディスプレイテレビのチャンネル表示スイッチのON／OFFを反転します。これも、HuBASICではサポートされておらず、テレビに付属のスイッチによらねばなりません。

〔実験3〕

サブCPUからデータを受け取る実験です。ここでは、1番タイマーの設定状況を6バイトデータとして受け取ります。あらかじめ、1番タイマーにデータをセットしておいてから実験するとよいでしょう。現在時刻や月・日等は、HuBASICがサポートしていますが、個々のTVタイマー（1番〜7番）からのデータをプログラム中で直接得ようとしたら、このようなルーチンを経由しなくてはなりません。

以下に、3つの実験リストを掲げます。

リスト　　　実験 1, 2, 3

```
0000                  ;************************************
0001                  ;  Example 1 of sub-CPU control
0002                  ;************************************
0003                  ;
0004                  ;
0005                          ORG   0E000H
0006                  ;
0007                  ;
0008 E000 3ED5                LD    A,0D5H          ;command of TIMER5 setting
0009 E002 0606                LD    B,6             ;6 bytes data
0010 E004 1110E0              LD    DE,DATA1        ;data start address
0011                  ;
0012 E007 CD00F0              CALL  0F000H          ;subroutine of OUT SUB-CPU
0013                  ;
0014 E00A C30010              JP    1000H           ;goto MONITOR
0015                  ;
0016                  ;
0017                          ORG   0E010H
0018                  ;
0019                  ;-----------------
0020                  ; DATA for output
0021                  ;-----------------
0022                  ;
0023 E010 40          DATA1:  DB    40H             ;TIMER set
0024 E011 97                  DB    97H             ;ON CH 8
0025 E012 34                  DB    34H             ;minute = 34
0026 E013 12                  DB    12H             ;hour   = 12
0027 E014 A1                  DB    0A1H            ;month  = 10 (A) , MON (1)
0028 E015 15                  DB    15H             ;day    = 15
0029                  ;
0030                  ;
0031                  ;************************************
0032                  ;  Example 2 of sub-CPU control
0033                  ;************************************
0034                  ;
0035                  ;
0036                          ORG   0E100H
```

```
0037                    ;
0038                    ;
0039 E100 3EE7         ;        LD    A,0E7H         ;command of TV control
0040 E102 0601         ;        LD    B,1            ;1 byte data
0041 E104 1110E1       ;        LD    DE,DATA2       ;data start address
0042                    ;
0043 E107 CD00F0       ;        CALL  0F000H         ;subroutine of OUT SUB-CPU
0044                    ;
0045 E10A C30010       ;        JP    1000H          ;goto MONITOR
0046                    ;
0047                    ;
0048                            ORG   0E110H
0049                    ;
0050                    ;-----------------
0051                    ; DATA for output
0052                    ;-----------------
0053                    ;
0054 E110 09            DATA2:  DB    09H            ;CH call ON/OFF reverse
0055                    ;
0056                    ;
0057                    ;*********************************
0058                    ;  Example 3 of sub-CPU control
0059                    ;*********************************
0060                    ;
0061                    ;
0062                            ORG   0E200H
0063                    ;
0064                    ;
0065 E200 3ED9                 LD    A,0D9H         ;command of TIMER1 read
0066 E202 0606                 LD    B,6            ;6 bytes data
0067 E204 1110E2               LD    DE,DATA3       ;data store address
0068                    ;
0069 E207 CD00F1               CALL  0F100H         ;subroutine of IN SUB-CPU
0070                    ;
0071 E20A C30010               JP    1000H          ;goto MONITOR
0072                    ;
0073                    ;
0074                            ORG   0E210H
0075                    ;
0076                    ;
0077                    ;-----------------
0078                    ; DATA store area
0079                    ;-----------------
0080                    ;
0081 E210               DATA3:  DS    6
0082                    ;
0083                    ;
0084                    ;
0085 E216                       END
```

6－7　HuBASICの関連内部ルーチン

前節において提示したサブCPUとのデータ入出力ルーチンは、読者の皆さんが、サブCPUの機能について理解を深めることを目的として作りましたので、レジスタの使い方など無駄もあろうかと思います。実用上は、HuBASICモニター内に用意されているサブルーチンを利用するとよいでしょう。

名　　称	サブCPUとのデータ入出力
アドレス	0023H（あるいは0E07H）
入力レジスタ	Aレジスタ　＝サブCPUへの入力・出力コマンド DEレジスタ＝データ格納先頭アドレス
機　　能	入力コマンド、出力コマンドに応じて、DEをポインタとし必要なバイト数だけ、サブCPUとデータの授受を行なう。

前節で紹介したような出力、入力の2つのルーチンが1つにまとめられているのと、また、コマンドを解析して必要なバイト数を割り出してくれるので、実用上は大変に便利です。

たとえば、HuBASICのモニター内に次のような処理があります。

```
LD      A,0EDH
LD      DE,1498H
CALL    0023H
```

読者の皆さんは、もうおわかりと思いますが、これはサブCPUから現在の「年・月・日」を得て、モニターのワークエリア（1498H番地～）に書き込む処理で、カセットにSAVEする時の日付をつけるのに用いられます。

第7章　キー入力割り込み

7－1　割り込みとは何か

CPUは、メモリーに格納されているプログラムを、通常小さい番地から大きい番地へと順に実行していきますが、JP命令や、サブルーチンを呼び出すCALL命令に出会うと指定番地に制御を移します。これはあくまで、プログラム（ソフトウェア）による流れの変え方ですね。

これに対して、キーボード等の周辺機器（ハードウェア）によりプログラムの流れを変える方法を**割り込み**（interrupt＝インタラプト）といいます。

X 1においては、キーボードからのキー入力を割り込みにより処理するよう設計がなされています（キー入力割り込み）。次の簡単なBASICプログラムを実行して、キー入力割り込みを体験してみましょう。

《キー入力割り込みを体験する》

```
10 TIME=0
20 IF TIME=10 THEN END
30 GOTO 20
Ok
RUN
Ok
;SLGPOAKA;:F,AJGKJGBMSFGJDFKJALLAGKJALDF
JKA;:AADJGIKJAKLKJKADJkl
```

プログラムをRUNしたら、でたらめにキーを押し続けます。約10秒間でプログラムは終了しますが、それと同時に御覧の通り、押し続けたキーの文字が画面に表示されますね。

こういった現象は、皆さんも割合と多く出会っているのではないでしょうか。種明かしは次の通りです。

ここで見た例では、実行中のプログラムも、また入力したキーの情報も大した意味はありませんが、プログラム実行中に大切なキー入力をすることもあり得る訳です。一番よく起こるのは、パソコンをワープロやタイプライタ代わりに使っていて、キー入力者のスピードが速い時などでしょう。こんな時、CPUが前のキーの表示処理をしているからといって、次のキー入力を無視されたら困りますね。このような問題を解決するのに、もってこいの方法が割り込み処理なのです。すなわち、プログラム実行中であっても、キーが押されると、プログラムを中断して、割り込み処理ルーチンへ飛び、この中で、押されたキー

のコードを **先行入力バッファ** とよばれるメモリー上のエリアに積み上げ、元のプログラムに復帰して続行するという具合です。先の例は、割り込み処理ルーチンが先行入力バッファに積み上げておいた文字が表示されたものです。

キー入力割り込みの仕組み

実行中
の
プログラム

キーを押す

割り込み発生

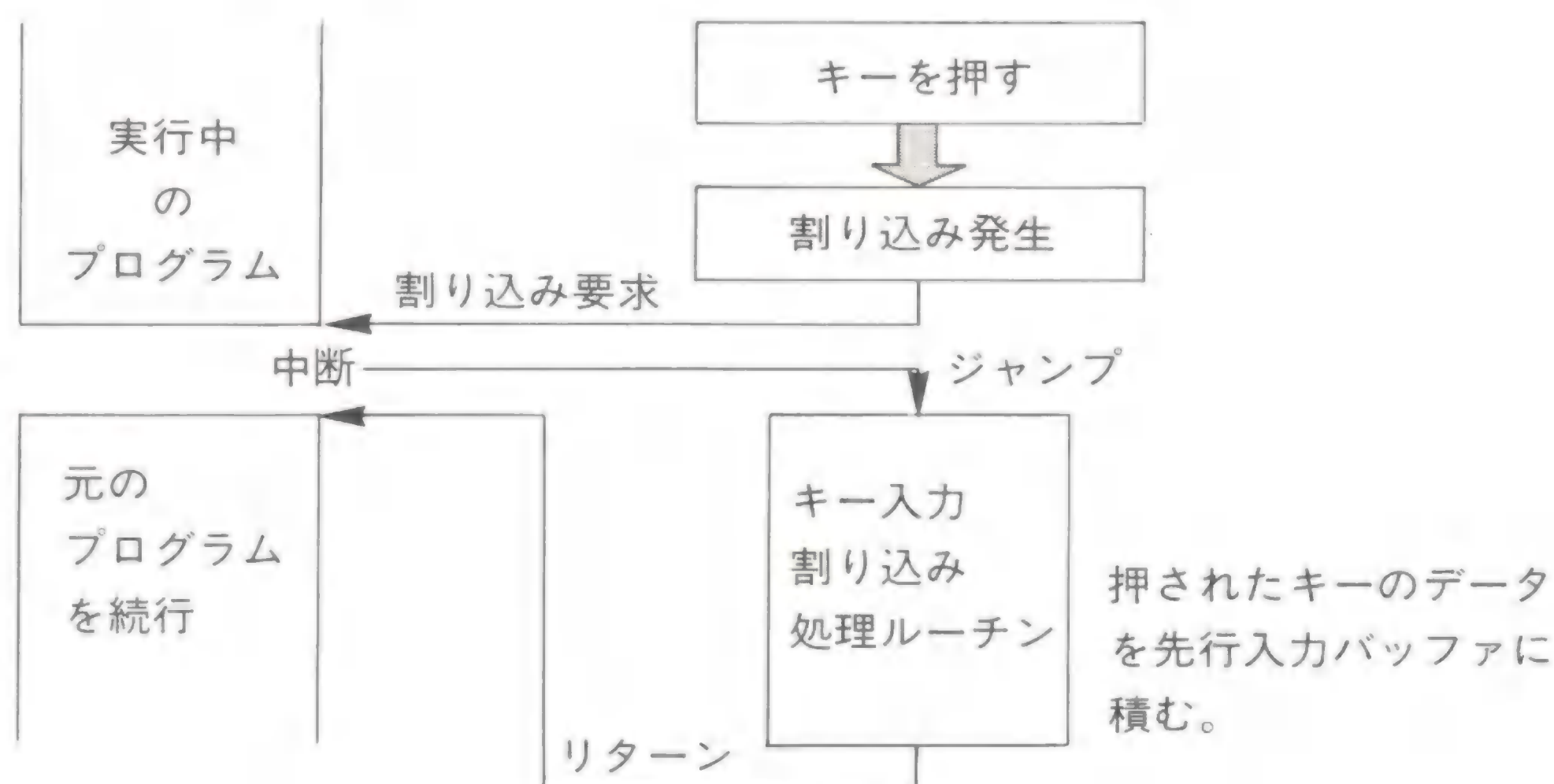
割り込み要求

中断

ジャンプ

元の
プログラム
を続行

キー入力
割り込み
処理ルーチン

押されたキーのデータ
を先行入力バッファに
積む。

リターン

処理の流れは、サブルーチンへのジャンプおよびメインルーチンへのリターンと酷似していますが、割り込みの場合はそれが、割り込み発生をするハードウェア（ここでは、キーボード）により起動される点で、サブルーチン呼び出しとは異なっています。

7－2　Z 80 A における割り込み

Z 80 A CPU は、

$\overline{\text{NMI}}$　＝　**マスク不能割り込み（ノンマスカブル・インタラプト）**

$\overline{\text{INT}}$　＝　**マスク可能割り込み**

という２種類の割り込み要求を受ける端子を持っています。

マスク不能割り込みは、プログラム中で割り込みを禁止（マスク）できない最優先の割り込みで、電源異常など緊急事態に対処するために使われます。

これに対して、**マスク可能な割り込み**は、プログラム中で**割り込み禁止命令**(DI＝Disable Interrupt）を実行すると、**割り込み許可命令**（EI＝Enable Interrupt）を実行するまで、割り込み要求があっても、受けつけなくすることができます。マスク可能割り込みには、モード０，１，２の３種類がありますが、X １のキー入力割り込みは、モード２を使用しています。

本章では、マスク不能割り込みとモード２のマスク可能割り込みについて必要な範囲で説明いたします。モード０，１も含め、より詳しい説明は、参考文献〔１〕,〔２〕,〔６〕などに譲ります。

7－3　マスク不能割り込み

Z 80 A CPU の $\overline{\text{NMI}}$ 端子が"L"レベルになると、CPU は現在実行中の命令が終了し次第、割り込み要求を受けつけ、0066 H 番地へジャンプします。

X 1の HuBASIC では、0066 H 番地から3バイトにわたってジャンプ命令が置かれ、システムを初期化するルーチンへ飛ぶように設計されています。

《HuBASIC でのマスク不能割り込み処理》

```
0066H番地……C3H ┐
0067H番地……FAH ├ JP 00FAH
0068H番地……00H ┘
```

X 1では、$\overline{\text{NMI}}$ 端子はリセット・スイッチと外部コネクタにつながれているので、リセット・スイッチを押すか、あるいは、接続したボードから $\overline{\text{NMI}}$ 信号を出した時に、マスク不能割り込みが起動され、システム初期化が実行されます。

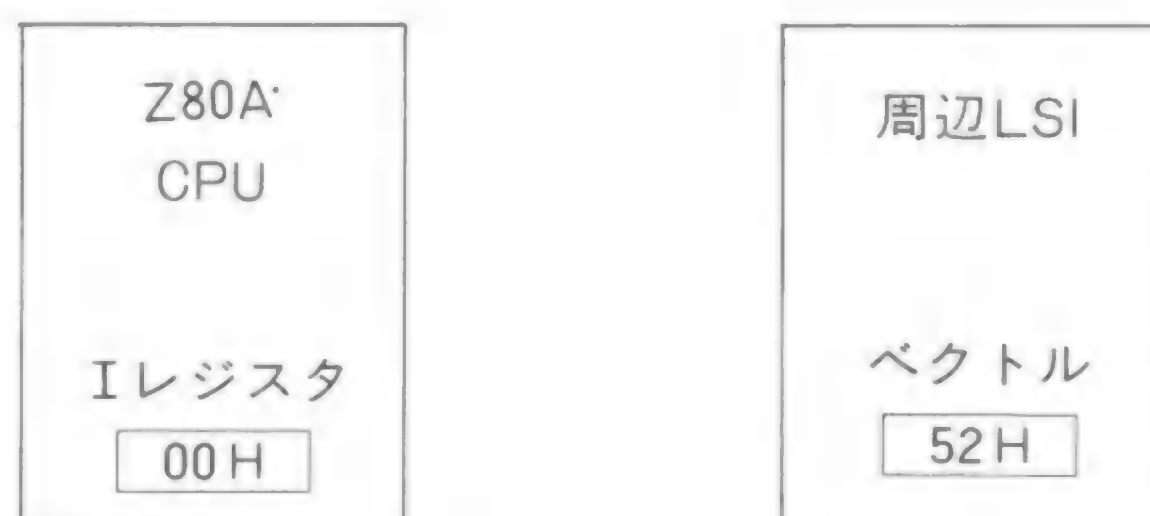
ユーザーのマシン語プログラムが「暴走」をした時、それが軽度なものなら、リセット・スイッチを押せば BASIC のコマンド・レベルへ戻れますが、これはマスク不能割り込みにより強制的に初期化をしたからなのです。もっとも、「暴走」によりこれらのルーチンが壊れている場合には、どうすることもできませんが。

以上が、X 1におけるマスク不能割り込みの使われ方です。

7－4　モード2の割り込みの仕組み

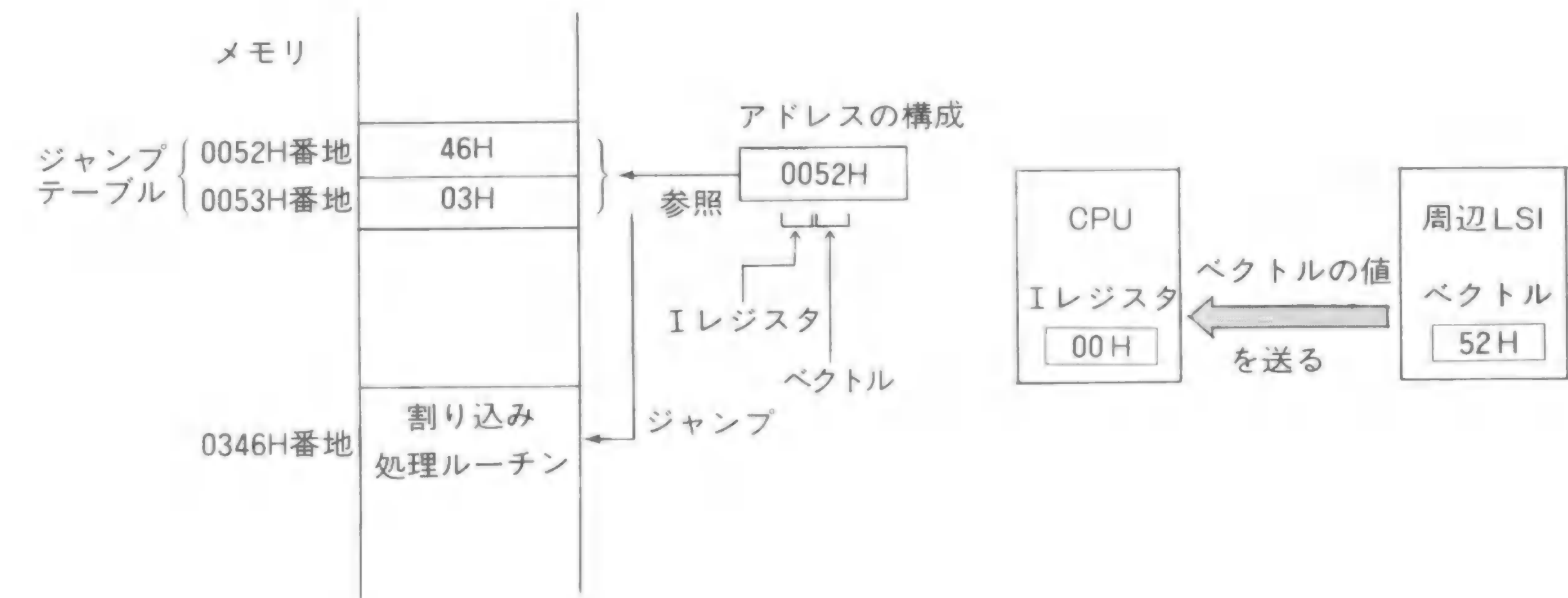
次にいよいよ Z 80 A CPU の特色の1つとなっている「モード2の(マスク可能)割り込み」について説明します。割り込み信号を発生するための周辺 LSI(ペリフェラル) を1つ考えましょう。

あらかじめ、周辺 LSI 内のレジスタに、**ベクトル (vector)** とよばれる1バイトデータを書き込んでおきます。また、Z 80 A CPU には、**インタラプト・レジスタ(Ｉレジスタ)** とよばれる8ビットのレジスタがありますが、ここにも1バイトデータを書いておきます。例えば次のように設定したとします。

Z80A CPU	周辺LSI
Iレジスタ	ベクトル
00H	52H

さて、プログラム内で「モード2の割り込み」を使うには、次の命令を実行しておかねばなりません。

ニーモニック	IM　2
マシンコード	ED　5E
機　　　能	割り込みモードを2に設定する。

以上の状況で、割り込み許可命令（EI）が実行され、周辺LSIから割り込み要求が来たとしましょう。ここからがモード2の割り込みの面白い所です。

CPUが割り込みを受けつけると、周辺LSIは書き込まれたベクトルの値をデータ・バスに出力します。CPUはベクトルを受けとり、これを下位8ビットとし、Iレジスタの内容を上位8ビットとするアドレスを参照します。そして、このアドレスと次のアドレスの2バイトに書かれている値を新しいアドレスとして、そこにジャンプするという仕組みです。

最初のうちは「何て面倒なことを！」と思われるかもしれませんが、このような手続きなら、ジャンプ・テーブルの値さえ書き換えれば容易に割り込み処理を変更することができる訳ですね。さらに、**デイジー・チェーン**といって、多くの周辺LSIを「芋づる式」に並べて割り込みに優先順位を決めたりするのにも便利にできているのですが、これらは実際にボードを作成して説明する方がよいと思いますので、モード2割り込み技法について、より詳しく知りたい方は文献〔2〕,〔3〕,〔6〕等を参照して下さい。

次節では、X1におけるモード2割り込みについて具体的に説明します。

7－5　X 1でのモード2割り込み

X 1のHuBASICでは、0052 H番地～0065 H番地が割り込みジャンプ・テーブルに当てられています。ジャンプ・テーブルは10個分設けられていますが、HuBASICでは、0052 H，0053 Hの1個分しか使用していません。

周辺LSIとしては、サブCPU　80 C 49が想定されていて、ベクトル52 Hが書き込まれます。CPU内のIレジスタには、00 Hが書き込まれます。こうして、キー入力があるとサブCPUから割り込み要求が出され、CPUは0052 H番地のジャンプ・テーブルを参照して、0346 H番地からの「キー入力割り込み処理ルーチン」へとジャンプしてゆきます。

次に、X 1でのモード2割り込み設定に関係する部分のHuBASIC逆アセンブル・リストを参考までに掲げておきます。

リスト　　HuBASICの割り込み設定

```
割り込みジャンプ・テーブル
    0052  4603    DB    46H,  03H    ; キー入力割り込み処理
    0054  D303    DB    D3H,  03H  ┐
    0056  D303    DB    D3H,  03H  │
    0058  D303    DB    D3H,  03H  │
    005A  D303    DB    D3H,  03H  │
    005C  D303    DB    D3H,  03H  ├ 未使用(ここへジャンプしても何もしない)
    005E  D303    DB    D3H,  03H  │
    0060  D303    DB    D3H,  03H  │
    0062  D303    DB    D3H,  03H  │
    0064  D303    DB    D3H,  03H  ┘
割り込みモード設定
    011B  ED5E    IM    2            ; 割り込みモード2に設定
    011D  214603  LD    HL, 0346H   ┐
    0120  225200  LD    (0052H), HL ├ ジャンプ・テーブルの設定
    0123  CD2D01  CALL  012DH       ┘
ベクトルとIレジスタ設定
    012D  215200  LD    HL, 0052H
    0130  7C      LD    A, H        ┐
    0131  ED47    LD    I,  A       ┘ Iレジスタを00Hに設定
    0133  3EE4    LD    A, 0E4H     ┐
    0135  CDFE0D  CALL  0DFEH       │
    0138  7D      LD    A, L        ├ サブCPUにベクトル52Hを書き込む。
    0139  C3540B  JP    0B54H       ┘
```

7－6　キー入力割り込み処理ルーチン

HuBASICでは、0346H～03D5H番地がキー入力割り込み処理の主ルーチンとなっています。このルーチンの入り口と出口の部分を逆アセンブルしたものが、次のリストです。

《キー入力割り込み処理ルーチン》

```
0346 F5          PUSH AF          03CF E1       POP  HL
0347 C5          PUSH BC          03D0 D1       POP  DE
0348 D5          PUSH DE          03D1 C1       POP  BC
0349 E5          PUSH HL          03D2 F1       POP  AF
034A AF          XOR  A           03D3 FB       EI
034B 32DA02      LD   (02DAH),A   03D4 ED4D     RETI
034E CD490B      CALL 0B49H
0351 57          LD   D,A
0352 CD490B      CALL 0B49H
0355 5F          LD   E,A
```

まず割り込み処理ルーチンの形の特徴を観察して下さい。最初の方でレジスタ類をスタックへ退避し、出口で元に戻し、全体としてレジスタの内容を保存していますね。割り込み処理は実行中のプログラムを中断して挿入される訳ですから、戻った時にレジスタが変化していたら困ります。割り込み処理ルーチンではこのように、レジスタの内容を保存するのが原則です（もっと徹底した例では、裏レジスタ、指標レジスタもすべて保存します）。

もう1つの特徴は出口で、EIとRETIが実行される点です。割り込みがかかると、自動的に割り込み禁止になりますから、EIで解除しておかないと次の割り込みがかからなくなります。また、RETIはReturn from Interruptを意味し、割り込み処理ルーチンからの特別なリターン命令です。

割り込み処理ルーチンは、このような形態上の特徴を持っているので、未知のプログラムを解析する際は手掛かりの1つになるでしょう。

さて、034EH番地と0352H番地の2箇所で、サブルーチン0B49Hがコールされていますが、ここがサブCPUからキー入力データを受けとる大切な部分です。

名　称	サブCPUからの1バイト入力
アドレス	0B49H
出力条件	Aレジスタ＝サブCPUからのデータ BCレジスタは保存。
機　能	サブCPU 80C49からの1バイト・データをAレジスタに受けとる。

キーボード内のサブCPU 80C48は、押されたキーの情報を2バイト・データにまとめて、本体内のサブCPU 80C49に直列データとして送信します。80C49は、これを受信すると並列データに変換して、メインCPUに割り込み要求を出します。

こうして、今問題にしている0346H番地からの割り込み処理ルーチンに飛んで来ます。ですから、このルーチン内で0B49Hが2回コールされているのは、サブCPUからの2バイトのキー情報を受け取ることを意味します。これらの2バイト・データの構成は次にようになっています。

《サブCPUからのキー・データ（1バイト目)》

キーボードからの入力状態を示すフラグの集まりです。

(MSB) ビット7	ビット6	ビット5	ビット4	ビット3	ビット2	ビット1	(LSB) ビット0
テンキー	キー入力	リピート機能	GRAPHキー	CAP LOCKキー	カナキー	SHIFTキー	CTRLキー
有0 無1	有0 無1	有0 無1	有0 無1	有0 無1	有0 無1	有0 無1	有0 無1

《サブCPUからのキー・データ（2バイト目)》

押されたキーのASCIIコードです。

キー入力割り込み処理ルーチンでは、これらのデータをワーク・エリアに格納した後(1バイト目は002FH番地に、2バイト目は002EH番地に入れられる)、1バイト目のフラグを見ながら処理をして、保存すべきデータは先行入力バッファに積み上げます。

〔注1〕

キー入力のときの「ブッ」という音をクリック音といいますが、03BEH〜03C7Hの処理でクリック音発生をしています。何とPSGから音を出しています。0E90H番地が、クリック音のON／OFFスイッチとして参照されます。HuBASICの命令で、CLICK OFFすると、このスイッチが00Hになります。

〔注2〕

カセットやテレビの制御キー（次節参照）などは、先行入力バッファには積まれません。

7−7 特殊キーのコードを見る

前節で、キー・データをサブCPUから受け取る仕組みを説明しましたが、これを利用して実験を試みてみましょう。

X1には、テレビとカセットの制御キーがキー・ボードに付いています（ただし、X1Cにはテレビ用のキーがなく、X1Dにはカセット用のキーがありません)。驚くべきことに、これらの特殊キーにも（ASCII）コードが決まっているのです。

ただ、通常のBASIC命令では、特殊キーのコードを調べることはできませんが、マシン語を利用し、押されたキーの2バイト情報をサブCPUから直接受け取れば、コードを見ることができます。

リスト　　キーのコードを見る

```
100 REM **************************
110 REM *                        *
120 REM *    ｷｰ ﾉ ｺｰﾄﾞ ｦ ﾐﾙ      *
130 REM *                        *
140 REM *                        *
150 REM *     by Y.Shimizu       *
160 REM *                        *
170 REM *     1984.6.26 TUE      *
180 REM *                        *
190 REM **************************
200 '
210 CLEAR &HE000
220 WIDTH 40 : INIT : CLS
230 CLICK OFF
240 '
250 REM *** ｶﾞﾒﾝ ﾂﾞｸﾘ ***
260 '
270 PRINT "KEY  code    status"
280 PRINT
290 PRINT "                TPRGLKSC"
300 '
310 REM *** ﾏｼﾝｺﾞ ｶｷｺﾐ ***
320 '
330 ADR=&HE000
340 '
350 READ MC$
360 IF MC$="END" THEN 520
370 MC=VAL("&H"+MC$)
380 POKE ADR,MC
390 ADR=ADR+1
400 GOTO 350
410 '
420 REM *** ﾏｼﾝｺﾞ ﾃﾞｰﾀ ***
430 '
440 DATA 11,10,E0         : REM  LD   DE,0E010H   ;work  ( 2 bytes )
450 DATA 3E,E6            : REM  LD   A,0E6H
460 DATA CD,23,00         : REM  CALL 0023H
470 DATA C9               : REM  RET
480 DATA END,             : REM  data end mark
490 '
500 REM *** ﾒｲﾝ･ﾙｰﾌﾟ *******************************************
510 '
520 CALL &HE000
530 STAT=PEEK(&HE010) : CODE=PEEK(&HE011)
540 LOCATE 0,4
550 PRINT#0 CHR$(CODE)+"    ";RIGHT$("0"+HEX$(CODE),2)+"H     ";
560 PRINT RIGHT$("0000000"+BIN$(STAT),8)
570 GOTO 520
580 '
590 REM ********************************************************
```

サブCPUにコマンドE6Hを出力して、2バイトのキー情報を得ます（450行～460行）。システムサブルーチン0023Hを利用しました（サブCPUの章を参照）。

〔注〕ゲーム中などで単に押されたキーのASCIIコードが欲しい時には、I／Oポートの1900H番地を読むだけで済みます。

さて、上のプログラムを実行して調べると、マニュアル付属の「ASCIIコード表」に掲載されているキーのコード以外に、次のようなことがわかります。

《マニュアルに記載されていない特殊キーのコード》

キーの名称	コード	キャラクタ	キーの機能
COMPUTER／TV	E8H	×	コンピュータ・テレビ切り替え
VOLUME ∧	E1H	○	テレビの音量を上げる。
VOLUME ∨	E2H	♠	テレビの音量を下げる。
CHANNEL ∧	EBH	■	テレビのチャンネルを順方向に進める。
CHANNEL ∨	ECH	■	テレビのチャンネルを逆方向に進める。
カセット・キー ■	C1H	チ	テープ走行の停止。
▶▶	C3H	テ	早送り。
◀◀	C4H	ト	巻き戻し。
SHIFT ＋ ■	C0H	タ	カセットのフタを開く。
SHIFT ＋ ▶▶	C5H	ナ	順方向頭出し(APSS＋1)。
SHIFT ＋ ◀◀	C6H	ニ	逆方向頭出し(APSS－1)。
ファンクションキーF1	71H	q	ファンクションキー 1番
F2	72H	r	2番
F3	73H	s	3番
F4	74H	t	4番
F5	75H	u	5番
SHIFT ＋ F1	76H	v	6番
SHIFT ＋ F2	77H	w	7番
SHIFT ＋ F3	78H	x	8番
SHIFT ＋ F4	79H	y	9番
SHIFT ＋ F5	7AH	z	10番

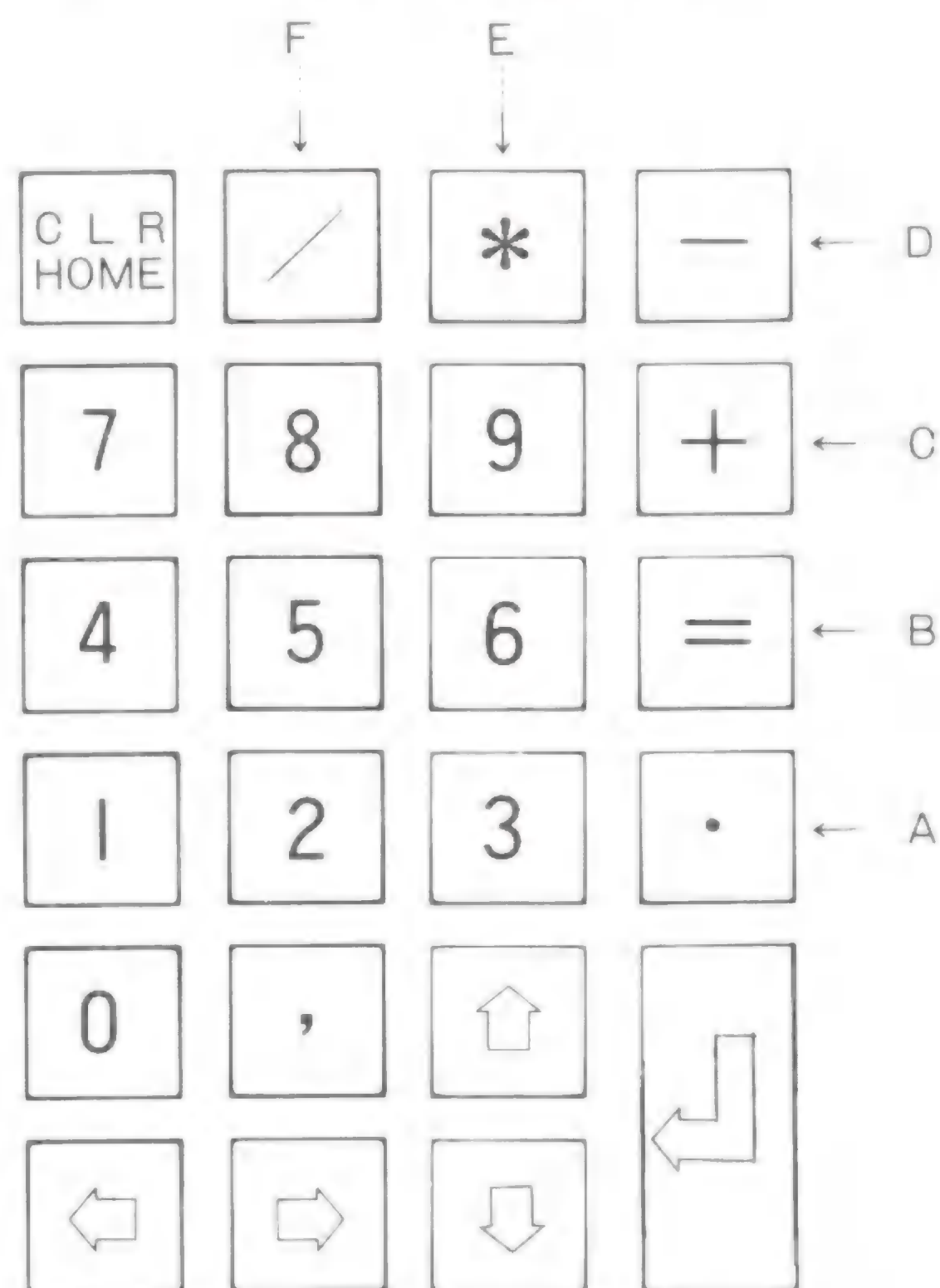
〔注〕 これらのキーは、すべて「テンキー」として扱われます。

7－8 テンキーを16進キーに変身させる

キー入力を扱った本章の最後に、非常に役に立つプログラムを紹介します。すなわち、テンキーを16進数入力用のキーに変えてしまうプログラムです。

X 1シリーズのテンキーは次のような配置になっていますが、⊡～◻のキーを16進数のA～F用に変えると、マシン語プログラムを雑誌等から入力する時、右手をテンキーに置いたまま入力できるようになり便利です。

図　　　　　テンキーの配置

	F	E		
CLR HOME	/	*	−	← D
7	8	9	+	← C
4	5	6	=	← B
1	2	3	.	← A
0	,	⇧	↵	
⇦	⇨	⇩		

さて、これを実現する方法ですが、ここではHuBASICのキー入力割り込み処理の内部構造を利用して設計しました。

HuBASICでは、03A6H番地から押されたキーがテンキーの場合の処理をしていて、03B4H～03B8Hで、[テンキー] あるいは [SHIFT] + [テンキー] の場合分けをします。ここに注目します。すなわち、JP命令によりキー入力割り込み処理の一部を外に引っぱり出して、16進キーの処理をした後、もとに戻すという方法を採用します。

16進キー処理の部分は次のように作ってみました。一応D000H番地から格納しましたが、「リロケータブル」なので、これ以外のアドレスにも配置することが可能です。

リスト　　　16進キー

```
0000                ;*************************
0001                ;*                       *
0002                ;*       HEXKEY          *
0003                ;*                       *
0004                ;*    by  Y.Shimizu      *
0005                ;*                       *
0006                ;*    1984.7.16 MON      *
0007                ;*                       *
0008                ;*************************
0009                ;
0010                ;
0011                        ORG   0D000H
0012                ;
0013 D000 7A        HEXKEY:LD    A,D               ;A = Key status
0014 D001 E602             AND   02H
0015 D003 2824             JR    Z,EXIT            ;if SHIFT then EXIT
0016                ;
0017 D005 7B        KYSCAN:LD    A,E               ;A = ASCII code
0018 D006 D5               PUSH DE
0019 D007 1646      HEXF:  LD    D,46H             ;D = 'F'
0020 D009 FE2F             CP    '/'
0021 D00B 2819             JR    Z,HEX
0022 D00D 15        HEXE:  DEC   D                 ;D = 'E'
0023 D00E FE2A             CP    '*'
0024 D010 2814             JR    Z,HEX
0025 D012 15        HEXD:  DEC   D                 ;D = 'D'
```

```
0026 D013 FE2D          CP    '-'
0027 D015 280F          JR    Z,HEX
0028 D017 15      HEXC: DEC   D              ;D = 'C'
0029 D018 FE2B          CP    '+'
0030 D01A 280A          JR    Z,HEX
0031 D01C 15      HEXB: DEC   D              ;D = 'B'
0032 D01D FE3D          CP    '='
0033 D01F 2805          JR    Z,HEX
0034 D021 15      HEXA: DEC   D              ;D = 'A'
0035 D022 FE2E          CP    '.'
0036 D024 2001          JR    NZ,ASCII
0037 D026 7A      HEX:  LD    A,D            ;A = 'A'-'F'
0038 D027 D1      ASCII: POP  DE
0039 D028 5F            LD    E,A            ;E = new ASCII
0040              ;
0041 D029 7A      EXIT: LD    A,D            ;preparation for system
0042 D02A E602          AND   02H
0043 D02C C3B703        JP    03B7H
0044              ;
0045 D02F               END
```

さて、問題はこのルーチンの実行法です。次のようにメモリー変更をすればよいのですが、この部分がキー入力割り込み処理の一部なので、モニターの＊M コマンドで変更すると、その瞬間、キー入力を受けつけなくなってしまいます。

《変更点》

アドレス	元のコード		変更コード
03 B 4	7 A	→	C 3
03 B 5	E 6	→	00
03 B 6	02	→	D 0

ではどうするかというと、プログラムでここを書き替えればよいのです。例えば、HEX-KEY プログラムのすぐ後に次のようなプログラムを入力します。

≪HEXKEY 起動プログラム≫

アドレス	コ ー ド	ニーモニック
D040	21 B4 03	LD HL, 03B4H
D043	3E C3	LD A, 0C3H
D045	77	LD (HL), A
D046	23	INC HL
D047	3E 00	LD A, 00H
D049	77	LD (HL), A
D04A	23	INC HL
D04B	3E D0	LD A, 0D0H
D04D	77	LD (HL), A
D04E	C3 00 10	JP 1000H

＊G D 040 により「HEXKEY 起動プログラム」を実行すると、以後 16 進キー入力が可能となります。テンキーを元に戻したければ、同様のプログラムにより、03 B 4 H～03 B 6 H 番地の内容を元に戻せばよい訳ですね。

また、HEXKEY プログラムを別のアドレスに配置した時は、03 B 4 H～03 B 6 H 番地のジャンプ先を、格納アドレスの先頭に設定して下さい。

第8章　カセット・データ・レコーダ

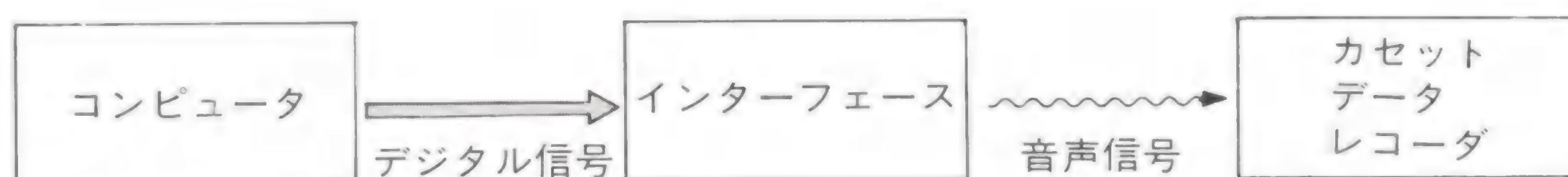
8－1　ボー・レートとは

最近では、パーソナル・コンピュータの補助記憶装置として、フロッピー・ディスクの普及が目覚ましいようですが、まだまだ、手軽で安価なカセット・テープの魅力も捨てがたい所があります。

Ｘ１およびＸ１Ｃには、補助記憶装置として、カセット・データ・レコーダが内蔵されていて、通常のカセット・テープにプログラムやデータを保存しておけるようになっています。本章では、この内蔵カセットの制御方法を紹介します。

まず最初に知らなくてはならないのは、**ボー・レート**という用語です。コンピュータが、メモリー（主記憶装置）内のプログラムやデータを補助記憶装置に移すこと（セーブ）、あるいはその逆に補助記憶装置からメモリーに読み込むこと（ロード）は、広い意味で、通信の問題と考えることができます。補助記憶装置としてカセット・データ・レコーダを用い、媒体にカセット・テープを想定する時には、メモリー内の各データは、ビット毎に一定の方法で音声信号に変えられ（変調）、テープに「録音」されることになります。

コンピュータ ⇒ インターフェース 〜〜〜▸ カセット データ レコーダ

デジタル信号　　音声信号

図のように、コンピュータとカセット・データ・レコーダは、インターフェースを介して、互いに「通信」するのだと考えることができます。

さて、通信において、まず大切なのは、その通信路あるいは転送方式が１秒間に何ビットの転送速度を持つかということでしょう。

これは速ければ速いほど能率は上がりますが、機械にはそれだけ信頼性の高さが要求されることになりますね。ここに言う所の転送速度を **ボー・レート**（baud rate） とよび、その単位である「１秒間に転送できるビット数」を **ボー**（baud） とよびます。

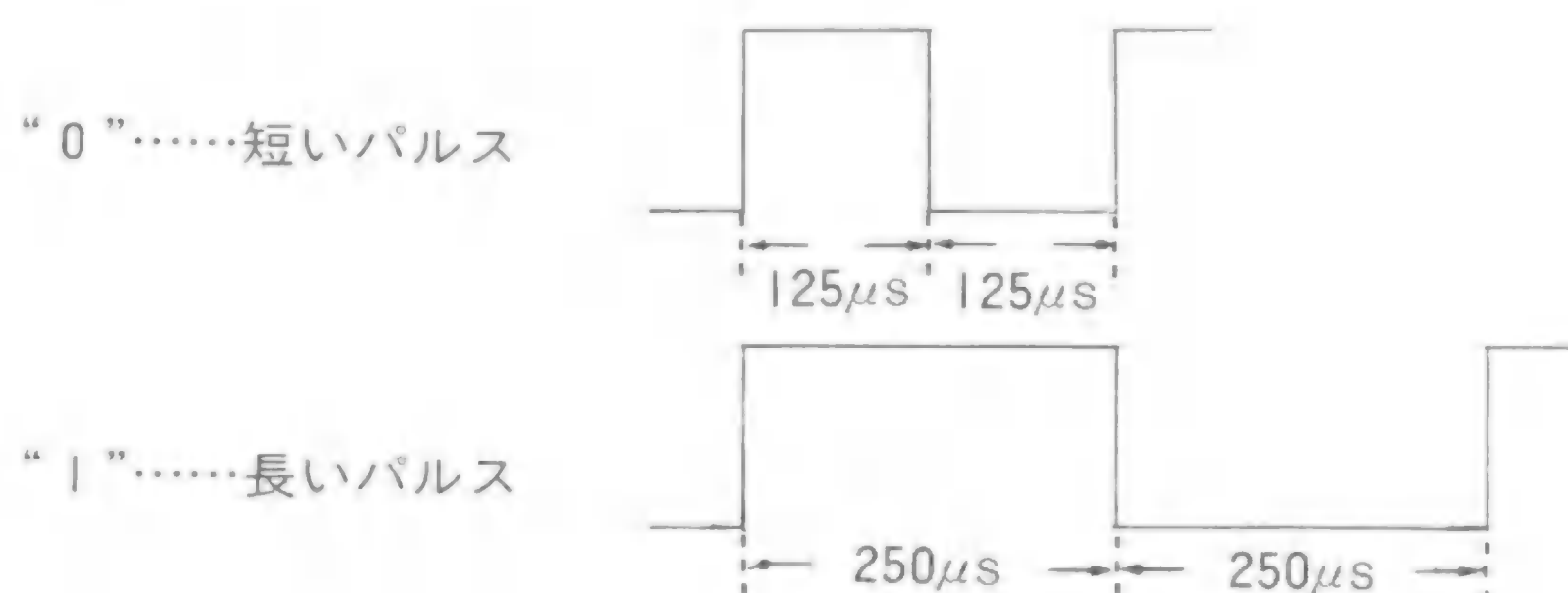
たとえば、PC-8001などはオーディオ用のカセット・デッキなども想定しているために、ボー・レートは600ボー、1200ボーのあたりに抑えてあります。これに対して、シャープのMZシリーズでは、クリーン設計のためシステム・プログラムを読み込む必要から、カセットを内蔵式にして信頼性を挙げ、ボー・レートを高く設定する努力がされてきたようです。たとえばMZ-2000では2000ボーに設定されています。

X 1シリーズは、MZシリーズとは異なる系列に属するパソコンですが、やはりその伝統を受け継いで、カセットのボー・レートは最高速のクラスに属する2700ボーに設定されています。カセットのHuBASICを2分何十秒もかけてロードする時、待ち遠しい思いをしますが、これは**2700ボー**の高速ロードだからこの時間で済んでいるのであって、他機種のボー・レートで40Kバイト以上もあるHuBASICを読み込むとしたら、恐らく実用にはならなかったことでしょう。

8－2　2700ボーの意味

X 1のボー・レートが2700ボーというのは、1秒間に2700ビットを転送することのできる速度を意味していました。本節ではさらに、コンピュータのデジタル信号をどのように音声信号に変調するかを考えてみましょう。

オーディオ用のカセット・デッキでは、アナログ信号を扱いますから、大きな音は大きな電圧、小さな音は小さな電圧というように音量に応じた電圧変化としてテープに録音されますが、コンピュータの補助記憶装置としてのカセット・データ・レコーダでは、デジタル信号が相手ですから、電圧レベルは単純に2レベルあれば足りることになりますね。これを"H"レベル、"L"レベルとよぶことにしましょう。

さて、このように単純な電圧パターンで、"0"と"1"のデータを表わすには様々な方法が考えられます(興味のある読者は、文献〔15〕などを見ると面白いでしょう)。X 1においては、2種類のパルスを用意し、短いパルスで"0"、長いパルスで"1"を表わす方法(パルス幅変調＝PWMといいます)を用いています(下図参照)。

"0"を表わす短いパルスでは、山である"H"レベルが 125 μs の間続き、谷である"L"レベルが 125 μs 間続きます。 μs はマイクロ・セカンドと読み、1秒の100万分の1を表わす単位です($1\mu s=10^{-6}s$)。"1"に対応する長いパルスは、長さが倍になっています。波を表わすのに、よく周波数という用語を使いますが、これは1秒間に何個分の波が入るかを表わす（単位 Hz）ものですから、上のパルスを周波数で示すと次のようになります。

短パルス……1波形（山と谷）の時間$=125\mu s\times2=250\times10^{-6}s$
　　　　　　周波数$=1/(250\times10^{-6}s)=4000$ Hz
長パルス……1波形の時間$=250\mu s\times2=500\times10^{-6}s$
　　　　　　周波数$=1/(500\times10^{-6}s)=2000$ Hz

五線譜での普通のラの音が 440 Hz ですから、"0"、"1"ともに随分と高い音で録音されることになります。コンピュータのプログラムテープをオーディオ用のデッキで聴くと、「ピーピーガーガー」と聞こえるのは、そのためなのです。

さて、コンピュータから送られるデータ信号内には"0"と"1"が様々なパターンで並んでいるはずですが、仮りに、"0"と"1"が均等に分布していると仮定すると、短パルスと長パルスの平均が、1ビットの平均パルス長となるはずです。

$$平均パルス1波形の時間=\frac{125\mu s\times2+250\mu s\times2}{2}=375\mu s$$

$$平均パルスの周波数=\frac{1}{375\times10^{-6}s}\fallingdotseq2666.7\mathrm{Hz}$$

もし長短を平均したパルスを考えるなら、その周波数は上のようになるはずです。そして、平均パルスは1ビットに対応するものでしたから、周波数（1秒間のパルス個数）がボー・レートに相当しますね。X 1のボー・レートが 2700 ボーということの正確な内容はこのようなものです（**平均ボー・レート**という）。従って、データの中に"0"が多ければ、ボー・レートは 2700 ボーより高めになり、"1"が多ければ低めになります。パルスの長短でデジタル信号を表わす方式である以上、これは仕方のないことですね。

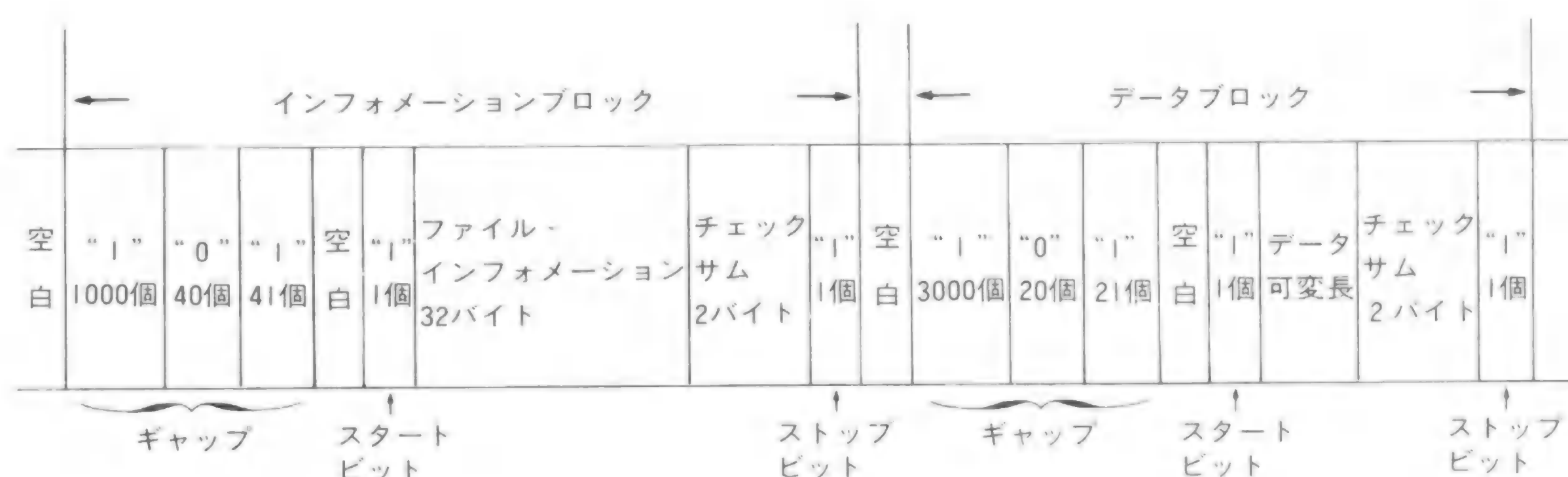
8－3　テープ・フォーマット

前節までに述べてきたボー・レートの問題は、テープにどのくらいの密度で情報を積め込むかということでした。次に大切なのは、データ列を格納していく形式についての問題です。たとえば、あるプログラムがテープに記録してあるとして、それを再生して読み込むとき、どこを読み込みの最初にするか等は、キチンと決めておかねばなりません。このようなテープへの記録形式をテープの**フォーマット**（format）とよびます。

フォーマットは、パーソナル・コンピュータの機種によってまちまちに設定されています。たとえ、ボー・レートを共通にできても、普通の方法ではPC-8001用のプログラム・テープをX 1で読めないのは、このフォーマットの違いによる所が大です。また、シャープのようにクリーン設計の場合、走らせるBASICによっても、フォーマットが異なります。たとえば、S-BASICの系列と、HuBASICの系列ではフォーマットが異なっています。ただし、X 1では後述するように、ボー・レートの設定やフォーマットはすべてシステムプログラム(HuBASIC)によりソフトウェアで管理されていますから、自分で変更することが可能です。X 1をMZシリーズやPC-8001に変身させるシステムコンバーターが市販されていますが、その中ではこのような処理をして、テープを読めるようにしているのです。

本節では、X 1シリーズに標準装備されるHuBASICのテープ・フォーマットについて説明します。

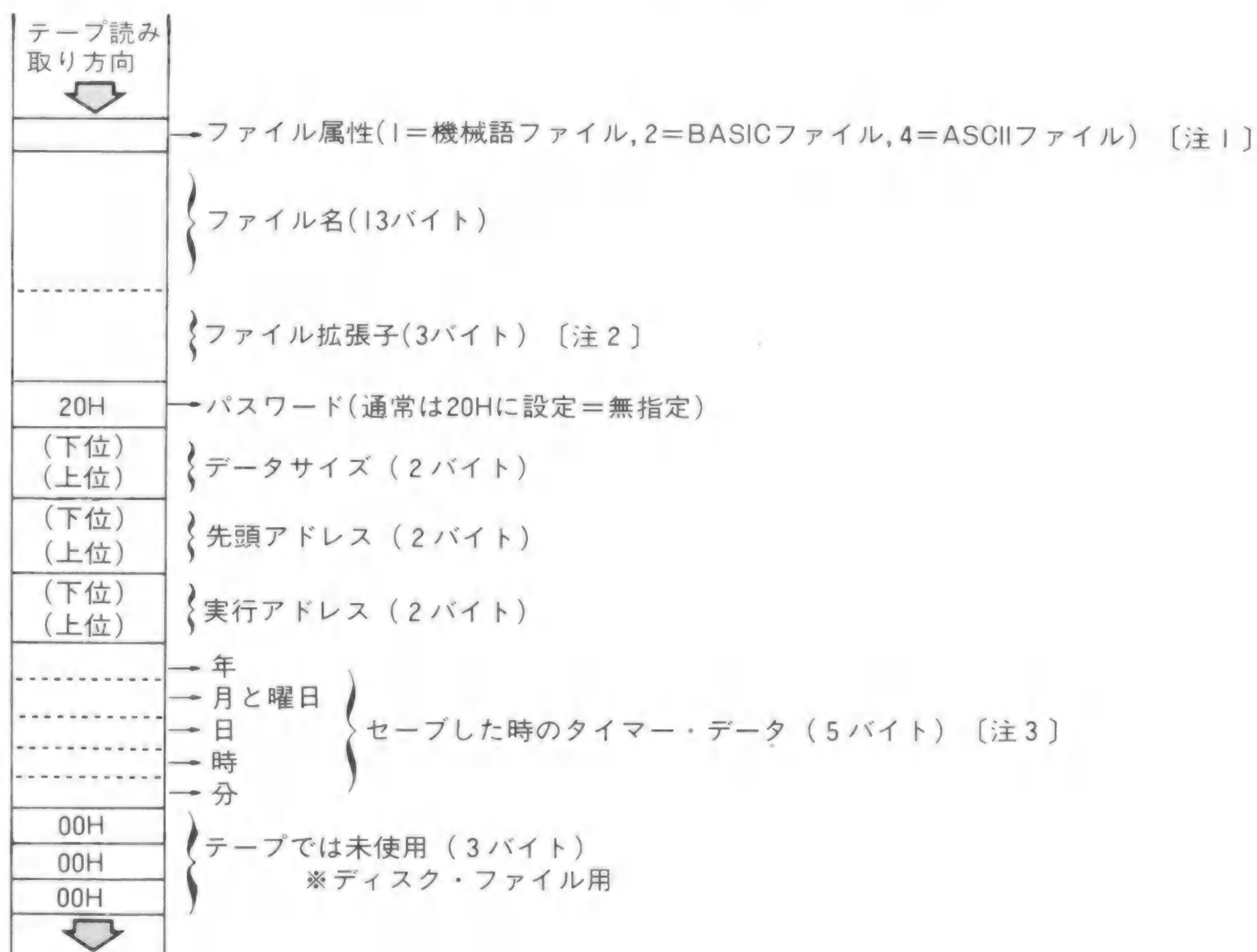
図　X 1　HuBASICのテープ・フォーマット

	←			インフォメーションブロック				→		←			データブロック				→
空白	"1" 1000個	"0" 40個	"1" 41個	空白	"1" 1個	ファイル・インフォメーション 32バイト	チェックサム 2バイト	"1" 1個	空白	"1" 3000個	"0" 20個	"1" 21個	空白	"1" 1個	データ 可変長	チェックサム 2バイト	"1" 1個
	ギャップ				スタートビット			ストップビット		ギャップ				スタートビット			ストップビット

HuBASICの扱うすべてのテープ・ファイルは、インフォメーション・ブロック、データ・ブロックという2つの大きな単位から構成されています。テープ上には、いくつかのファイルが1列に並び、各ファイルは2つのブロックに分かれていますから、各部分の区切りをきちんとしておく必要があります。これを**ギャップ**(gap＝間隙)とよびます。X 1では、特定個数ずつの"1"、"0"、"1"の列がギャップとして用いられています。

インフォメーション・ブロックは、そのファイル自体についての情報を記録しておく部分で、本体は、32バイトのデータからなります。

図　　インフォメーション・ブロック32バイト内訳

テープ読み取り方向

	ファイル属性(1＝機械語ファイル, 2＝BASICファイル, 4＝ASCIIファイル)〔注1〕
	ファイル名(13バイト)
	ファイル拡張子(3バイト)〔注2〕
20H	パスワード(通常は20Hに設定＝無指定)
(下位)(上位)	データサイズ(2バイト)
(下位)(上位)	先頭アドレス(2バイト)
(下位)(上位)	実行アドレス(2バイト)
	年／月と曜日／日／時／分　セーブした時のタイマー・データ(5バイト)〔注3〕
00H	テープでは未使用(3バイト)
00H	※ディスク・ファイル用
00H	

〔注1〕 files をとった時に、ファイル名の先頭に表示される文字がファイル属性です。
Bin＝機械語ファイル(バイナリー・ファイル)
Bas＝BASIC ファイル
Asc＝ASCII 形式でセーブされた BASIC ファイル、および、データ・ファイル

〔注2〕たとえば、“TEST. bas” のようなファイル名をつけると、ピリオド後の3文字 bas がファイル拡張子となります。同名のファイルを区別する時に用います。なお、IPL 起動型のプログラムでは、拡張子を Sys としなくてはなりません。

〔注3〕タイマー・データは次の通り。
年　…………年を表わす西暦下2桁のBCD(2進化10進数)コード。(例：1984年なら84Hとする。)
月と曜日……上位4ビットが月を16進表示したもの。
下位4ビットがSUN, MON…SATを0～6にあてたもの。(例：10月で土曜日ならA6Hとする。)
日、時、分…各々、BCDコードにて表わす。

こうして、たとえば1984年6月23日 SAT 18：30 にセーブされたファイルのタイマー・データは、

84 H,66 H,23 H,18 H,30 H

という5バイトで表わされます（サブCPUの章を参照）。

ファイルインフォメーション本体32バイトのあとには、32バイト分のチェック・サム2バイトが記録されます。ただし、このチェック・サムは通常のマシン語プログラム入力時などに用いるサムではなく、32バイトのビット列中にある"1"の個数を合計したものです。たとえば、01 H，02 H，04 H，08 H，という4バイトのデータ列を考えると、通常のチェック・サムでは、0 FHですが、テープでのチェック・サムは、"1"を数えて4個ですから、04 Hとなります。

データ・ブロックは、記録されるデータのバイト数により長さは変わります。データ・ブロックの最後には、やはりチェック・サムが2バイト記録されます。

8－4　セーブ関係システム・サブルーチン

HuBASICのIOCS（入出力制御システム）内には、テープへのセーブ、テープからのロードを扱うサブルーチンが用意されています。解析するとわかりますが、これらのルーチンはかなり微妙にできているので、ボーレートやフォーマット変更などの特別の目的以外では、システム・サブルーチンをそのまま利用するのがよいと思います。そこで本節では、テープへのセーブに関係するサブルーチンの機能と使い方をまとめてみました。

名　　称	インフォメーション・ブロック　SAVE
アドレス	003BH（あるいは　0B75H）
入力レジスタ	HLレジスタ＝インフォメーション・データ先頭アドレス BCレジスタ＝インフォメーション・バイト数(20H)
機　　能	インフォメーション・ブロックのフォーマットに従って、ギャップ等を記録した後、HLレジスタの示す番地から32バイト分のインフォメーションをテープに書き込む。
注　　意	必ずBCレジスタに 20H＝32 をセットして呼び出すこと。

名　　称	データ・ブロック　SAVE
アドレス	003EH（あるいは　0B79H）
入力レジスタ	HLレジスタ＝データ先頭アドレス BCレジスタ＝データのバイト数
機　　能	データ・ブロックのフォーマットに従って、ギャップを記録した後、HLレジスタの示す番地から、BCレジスタのバイト数だけデータをテープに書き込む。

これらのルーチンから呼び出される下位のサブルーチンのうち主なものは次の通りです。解析に御利用下さい。

アドレス	機　　　　能
0BBDH	SAVE共通サブルーチン。HLをポインタとし、BCバイトだけ、データをテープに書き込み、末尾に2バイトのチェックサムを記録する。
0C7CH	テープにE6Hを800個書き込む。
0C8AH	Aレジスタの1バイト・データをテープに書き込む。ただし、1バイトの先頭にスタートビット"1"を加えて、計9ビット分を記録する。その際、チェックサムを更新する。
0CD6H	チェック・サム用バッファ(0EA1Hと0EA2H)をクリアする。
0CDFH	テープにギャップを書き込む。
0D15H	テープの走行をチェックする。
0D8AH	テープに短いパルスを書き込む("0"に対応)。
0DA5H	テープに長いパルスを書き込む("1"に対応)。
0DC7H	カセットのモーターを起動する。Aレジスタのビット0またはビット1が立っていればWRITEで、ビット0, 1がいずれも立ってなければREADでモーターをONする。
0DEAH	カセットを停止する。
0DF6H	テープの状態コードをサブCPUからAレジスタに受ける。

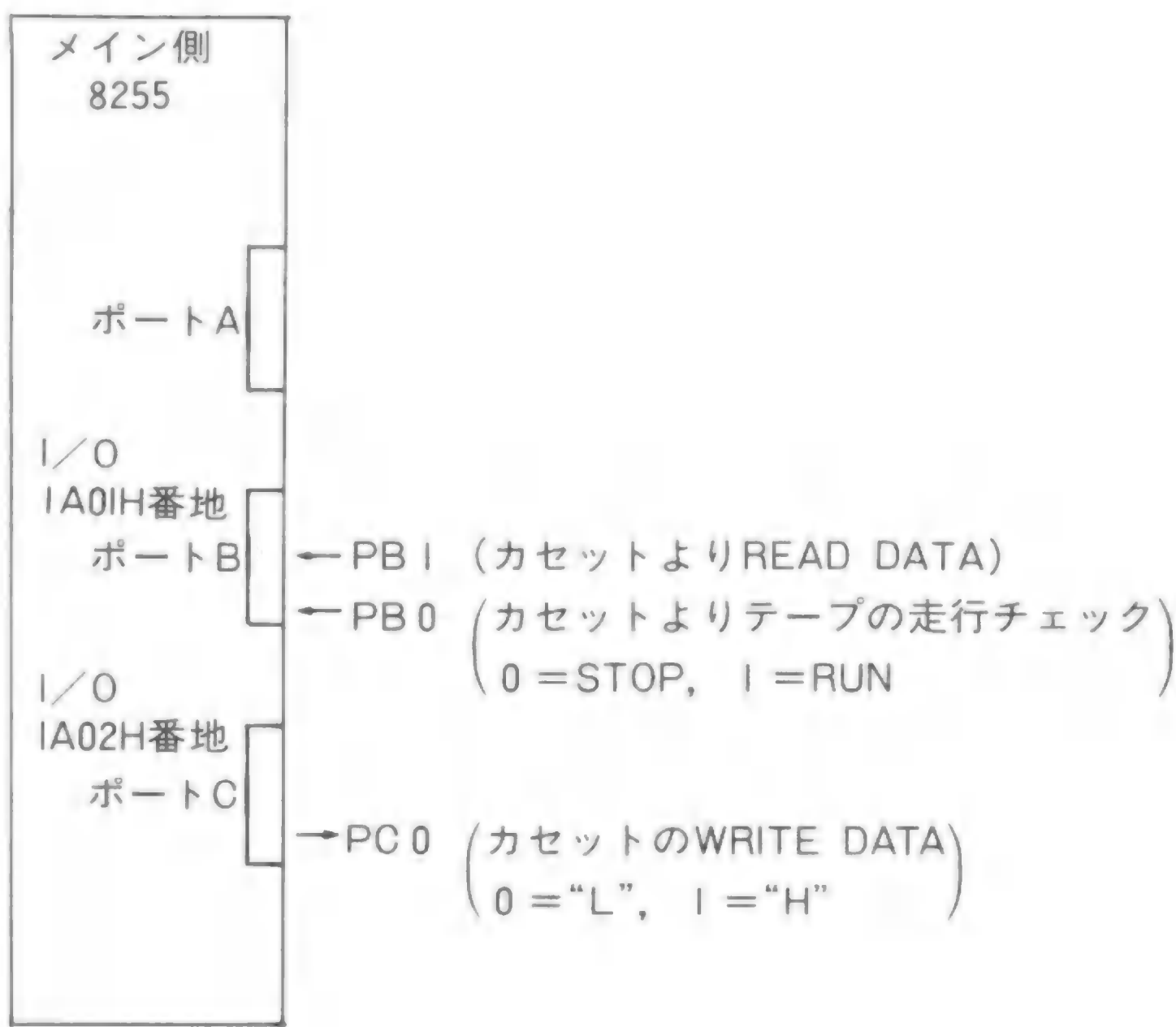
これらのサブルーチンで用いられる、カセット・データ・レコーダ関係のI／Oポートは次の通りです。

図　　カセット・データ・レコーダ関係I／Oポート

メイン側
8255

ポートA

I／O
1A01H番地
ポートB　←PB1 (カセットよりREAD DATA)
　　　　←PB0 (カセットよりテープの走行チェック 0＝STOP, 1＝RUN)

I／O
1A02H番地
ポートC　→PC0 (カセットのWRITE DATA 0＝"L", 1＝"H")

〔注1〕　カセット・テープの走行チェックは、I／Oポート1A01H番地のビット0を見ればよい。通常、

```
LD   A, 1AH
IN   A, (01H)
RRCA
```

により、ビット0をCyフラグにとり込んで判定する。

〔注2〕　テープにパルスを書き込むには、PC0を一定時間"H", "L"にすればよい。通常、8255ポートCのビット・モードを用いて、PC0をセット，リセットする。

```
LD   BC, 1A03H
LD   A, 01H
OUT  (C), A
```

により、PC0は"H"となる。

ボー・レートは、パルス書き込みルーチン（0D8AH，0DA5H）で設定されているので、各命令のクロック数をよく考えてパルス幅を調節すれば、ボー・レートの変更(SAVE時）が可能です。

8−5　ロード関係システム・サブルーチン

本節では、テープからのロードに関係するサブルーチンを紹介しましょう。

名　称	インフォメーション・ブロック　LOAD
アドレス	0041H（あるいは 0B9AH）
入力レジスタ	HLレジスタ＝インフォメーション・データ LOAD 先頭アドレス BCレジスタ＝インフォメーション・バイト数(20H)
機　能	インフォメーション・ブロックのフォーマットに従って、ギャップ等の確認の後、32バイト分のインフォメーションをテープから読みとり、HLレジスタの示す番地から、メモリーに格納する。
注　意	必ずBCレジスタに 20H＝32 をセットして呼び出すこと。

名　称	データ・ブロック　LOAD
アドレス	0044H（あるいは0B9EH）
入力レジスタ	HLレジスタ＝データLOAD先頭アドレス BCレジスタ＝データのバイト数
機　能	データ・ブロックのフォーマットに従って、ギャップを確認の後、BCレジスタのバイト数だけデータをテープから読み、HLレジスタの示すメモリー番地以降に格納する。
注　意	HL、BCは、通常インフォメーション・ブロックに記録されている値にセットする。

前節と同じく、これらのルーチンから呼び出される下位のサブルーチンの主なものを挙げておきます（前節と共通するものは省きました）。

アドレス	機能
0BF6H	LOAD共通サブルーチン。HLをポインタとし、BCバイトだけ、テープからデータを読みとり、メモリーに格納する。同時にチェック・サムをとり、テープに記録されているチェック・サムとの比較をする。
0CAAH	テープから1バイトデータを読みとる。ただし、各1バイトの先頭のスタート・ビット"1"は読みとばし、8ビット分について処理を行なう。その際、チェック・サムを更新する。
0D20H	ギャップをチェックする。
0D5CH	カセットのREAD DATA端子(I/Oポート1A01H番地のビット1)が"L"から"H"に立ち上がるのを検出する。
0D6DH	カセットの走行チェックをしながら、約8秒間費やす。
0DBFH	約170.5μsだけ時間を費やす。

テープからデータを読む時の基本は1ビットの判別ですが、HuBASICのフォーマットでは、パルスの立ち上がり(電圧レベルが"L"から"H"に変化)を検出してから、約184μs後に"L"レベルになっていれば短パルス（="0"）と判断し、"H"レベルであれば長パルス（="1"）と判断するようにしています。このように、長短パルス幅の中間の値に設定してあるのは、回転ムラで多少パルス間隔が変化しても、誤読をしないためと思われます。

←125μs→

短パルスでは、"L"レベルになっている。

← 約184μs →

長パルスでは、"H"レベルになっている。

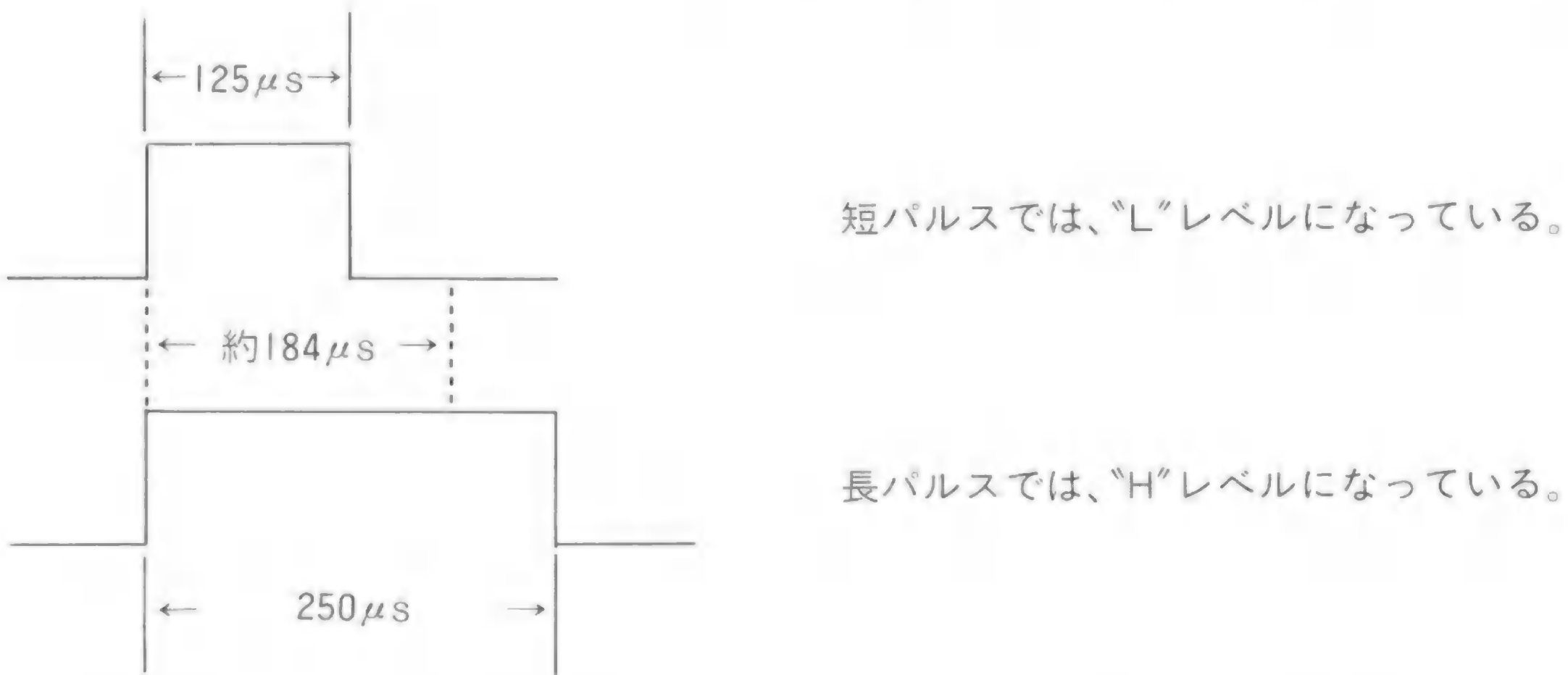
← 250μs →

参考までに、この部分の逆アセンブル・リストを掲げておきます。READ DATA信号を取り込んでから、次に信号を取り込むまでに735クロック経過しますが、X1のZ80A CPUは4MHzで動作していますから、1クロック=0.25μsであり、735クロックは時間にして、183.75μsに相当する訳ですね。前節で説明したボー・レート設定も、こうしたクロック計算をすると数字の意味がわかりますから、興味のある方は解析してみて下さい。

図　　クロック計算法

アドレス	マシン語	ニーモニック		クロック数	考え方
0CB3	110008	LD	DE, 0800H	10	D=8(1バイト=8ビット)、E=0(クリア)
0CB6	CD5C0D	CALL	0D5CH	17	
0D5C	ED78	IN	A, (C)	12	
0D5E	2F	CPL		4	
0D5F	0F	RRCA		4	
0D60	D8	RET	C	5(11)	
0D61	0F	RRCA		4	
0D62	30F8	JR	NC, 0D5CH	7(12)	
0D64	ED78	IN	A, (C)	12	→READ DATA 信号のサンプリング
0D66	2F	CPL		4	34クロック（0D66〜0D6C）
0D67	0F	RRCA		4	
0D68	D8	RET	C	5(11)	
0D69	0F	RRCA		4	
0D6A	38F8	JR	C, 0D64H	7(12)	
0D6C	C9	RET		10	
0CB9	3814	JR	C, 0CCFH	7(12)	24クロック（0CB9〜0CBB）
0CBB	CDBF0D	CALL	0DBFH	17	
0DBF	3E2E	LD	A, 2EH	7	665クロック（0DBF〜0DC6）
0DC1	00	NOP		4	
0DC2	3D	DEC	A	4	46回ループ（0DC2〜0DC3）
0DC3	C2C20D	JP	NZ, 0DC2H	10	
0DC6	C9	RET		10	735クロック（0D66〜0CBE）⇩ 183.75μs
0CBE	ED78	IN	A, (C)	12	→READ DATA 信号のサンプリング
0CC0	E602	AND	02H	7	ビット1をチェック
0CC2	2802	JR	Z, 0CC6H	7(12)	→この時点で初めて長パルス、短パルスを判別
0CC4	23	INC	HL	6	
0CC5	37	SCF		4	
0CC6	CB13	RL	E	8	パルス情報をEにとり込む。
0CC8	15	DEC	D	4	カウンタを1減らす。
0CC9	20EB	JR	NZ, 0CB6H	12(7)	

8－6　プログラムのオート・スタート

以上紹介したIOCS内のLOAD, SAVEサブルーチンを利用すれば、BASICのLOAD, SAVEコマンドでは不可能であった操作をすることができます。その典型的な例が、X1, X1Cのマニュアルに書かれている「BASICテープのコピー作成」プログラムです。本章での知識をもとにマシン語部分を解析すれば、何故にコピーが作成できるのか明瞭にわかることでしょう。

本節では、もう1つの例として、SAVEサブルーチンを利用したプログラムのオート・スタート（ロード完了と同時にプログラムを実行すること）法を紹介します。

リスト　AUTO START

```
0000                    ;*********************
0001                    ;
0002                    ;  AUTO START
0003                    ;
0004                    ;  by  Y.Shimizu
0005                    ;
0006                    ;  1984.6.24 SUN
0007                    ;
0008                    ;*********************
0009                    ;
0010                    ;==========      Hu monitor subroutines      ==============
0011                    ;
0012 003B               ISAVE: EQU  003BH          ;information block save
0013 003E               DSAVE: EQU  003EH          ;data block save
0014 0DEA               CMTSTP:EQU  0DEAH          ;cassette motor off
0015 0483               DSPMSG:EQU  0483H          ;message print ( pointer=DE,end=00
0016 04A7               CRLF:  EQU  04A7H          ;let new line
0017                    ;
0018 1000               MONHOT:EQU  1000H          ;monitor start
0019                    ;
0020                    ;=========================================================
0021                    ;
0022                    ;----------------------
0023                    ;  SAVE for AUTO START
0024                    ;----------------------
0025                    ;
0026                           ORG  0D000H
0027                    ;
0028 D000 2120D0        MAIN:  LD   HL,BUFFER      ;HL = buffer adrs for file inf.
0029 D003 012000               LD   BC,32          ;BC = bytes
0030 D006 CD3B00               CALL ISAVE          ;infomation block save
0031                    ;
0032 D009 2100E0               LD   HL,SAMPLE      ;HL = sample program top address
0033 D00C 010010               LD   BC,1000H       ;BC = bytes of sample
0034 D00F CD3E00               CALL DSAVE          ;data block save
0035 D012 CDEA0D               CALL CMTSTP         ;STOP CMT
0036                    ;
0037 D015 C30010               JP   MONHOT         ;goto monitor
0038                    ;
0039                    ;
0040                    ;
0041                           ORG  0D020H
0042                    ;
0043 D020 01            BUFFER:DB   01H            ;file attribute ( 01H=Binary )
0044 D021 4155544F             DB   'AUTO START    '   ;file name ( 13 bytes )
     D025 20535441
     D029 52542020
     D02D 20
0045 D02E 202020               DB   '   '          ;extention name ( 3 bytes )
0046 D031 20                   DB   20H            ;password
0047 D032 0010                 DW   1000H          ;file size
0048 D034 00F0                 DW   0F000H         ;load top address
0049 D036 00F0                 DW   0F000H         ;exec address
0050 D038 84                   DB   84H            ;timer data
0051 D039 60                   DB   60H            ;      84/6/24/SUN
0052 D03A 24                   DB   24H
0053 D03B 14                   DB   14H            ;      14:45
0054 D03C 45                   DB   45H
0055 D03D 000000               DB   00H,00H,00H    ;3 bytes 00H
0056                    ;
0057                    ;
0058                    ;
0059                    ;------------------------------------
0060                    ;  AUTO START SAMPLE ( E000H - EFFFH )
0061                    ;------------------------------------
0062                    ;
0063                           ORG  0E000H
0064                    ;
0065 E000 CDA704        SAMPLE:CALL CRLF           ;let new line on CRT
0066 E003 110FF0               LD   DE,MSG+1000H   ;DE = message data top address
0067 E006 CD8304               CALL DSPMSG         ;print message on CRT
0068 E009 CDEA0D               CALL CMTSTP         ;STOP CMT
0069 E00C C30010               JP   MONHOT         ;goto monitor
0070                    ;
0071 E00F 0D0A          MSG:   DB   0DH,0AH
0072 E011 436F6E67             DB   'Congratulation !'
     E015 72617475
     E019 6C617469
     E01D 6F6E2021
0073 E021 0D                   DB   0DH
```

```
0074 E022 57652073        DB   'We succeed AUTO START.'
     E026 75636365
     E02A 65642041
     E02E 55544F20
     E032 53544152
     E036 542E
0075 E038 0D0A            DB   0DH,0AH
0076 E03A 00              DB   00H               ;end mark for message
0077                 ;
0078                 ;
0079                 ;-------------------
0080                 ;  DATA for STACK
0081                 ;-------------------
0082                 ;
0083                      ORG  0EFF0H
0084                 ;
0085 EFF0 00F0            DW   0F000H            ;auto start address
0086 EFF2 00F0            DW   0F000H
0087 EFF4 00F0            DW   0F000H
0088 EFF6 00F0            DW   0F000H
0089 EFF8 00F0            DW   0F000H
0090 EFFA 00F0            DW   0F000H
0091 EFFC 00F0            DW   0F000H
0092 EFFE 00F0            DW   0F000H
0093                 ;
0094                 ;
0095 F000                 END
```

プログラムの実行に先立ち、内蔵カセットに新しいテープをセットしておいて下さい。準備ができたら、モニターより ＊G D000 でプログラムを走らせると、テープへのセーブが始まります。テープが止まったら、セーブした部分を頭出し (APSS－1) しておいて下さい。さて、モニターより ＊L で、今のセーブ部分をロードしますと、………めでたくオート・スタートしました。

オート・スタートできる理屈は次の通りです。オート・スタートさせたいプログラムをF000H番地から格納するように設計しておきます。そして、これを便宜的にE000H番地からメモリーに入れておきます。ところで、EFF0H番地からEFFFH番地のあたりを 00H, F0H というデータで満たしておくのを忘れぬようにして下さい。

さて、D000H番地からの SAVE プログラムは、E000H～EFFFH番地の1000Hバイト分を SAVE しますが、インフォメーション・ブロックを操作することで、ロード・アドレスを F000H からにしてしまいます。このように SAVE されたプログラムをモニターより、＊L でロードすると、F000H～FFFFH番地に格納されますが、モニターのスタック領域には、00H, F0H というデータ列が積まれてしまいます。こうして、＊L ルーチンが終って、スタックから戻り番地をポップすると、F000H がプログラムカウンタに入れられ、F000H から始まるプログラムにジャンプするという訳です。

原理は簡単ですが、この方法を応用すると、プログラムに「プロテクト」をかけることができます。オートスタートするプログラムの内容を、ボーレートやフォーマットを変更してロードするものにしておくと、HuBASIC からは通常のロードができなくなり、プログラム内容を秘密にすることが可能です。

このように、インフォメーション・ブロックやデータ・ブロックの LOAD, SAVE サブルーチンを応用することで、変則的な LOAD, SAVE を行なうことができる訳ですね。皆さんもいろいろ考えてみて下さい。

8－7　内蔵カセットを制御する

カセットのEJECT，テープの早送りなどは、内蔵カセットの場合、サブCPU 80C49 を通じて、プログラム中で行なうことができます。

サブCPUに、2バイトのコマンド列　E9H，x　を出力することで、内蔵カセットの制御ができます。 x は制御パラメータで、次の8種類の値が意味を持ちます。

《内蔵カセット　制御パラメータ》

x	名　称	機　能	対応するBASICコマンド
00H	EJECT	フタを開く。	EJECT ， CMT＝ 0
01H	STOP	カセット停止。	CSTOP， CMT＝ 1
02H	PLAY(READ)	再生。	CMT＝ 2
03H	FAST	早送り。	FAST ， CMT＝ 3
04H	REW	巻き戻し。	REW ， CMT＝ 4
05H	APSSI	順方向頭出し。	APSSI ， CMT＝ 5
06H	APSS-I	逆方向頭出し。	APSS-I， CMT＝ 6
0AH	RECORD(WRITE)	録音。	CMT＝10

HuBASICのIOCS内には、サブCPUにコマンド列 E9H，x を出力するサブルーチンが用意されています。

名　　称	内蔵カセット制御
ア ド レ ス	0DECH
入力レジスタ	Aレジスタ＝内蔵カセット制御パラメータ x
機　　能	サブCPUにコマンド列　E9H, x　を出力することで、内蔵カセットの制御を行なう。

よく用いるのは、カセットの停止を行なう制御で、Aレジスタに01Hをセットして、0DECH番地をコールします。このルーチンは、0DEAH番地で用意されています。前節の「AUTO START」プログラム中でも、このサブルーチンを利用しました。

8－8　テープの状態を調べる

HuBASICには、 CMT(x)〔ただし x は0，1，2のいずれか〕という特別な関数が用意されていて、カセット・テープの状態を調べることができます。

カセット・テープの状態は、サブCPU側8255のポートBに知らされてくるので、このポートを読み出せば調べられます。

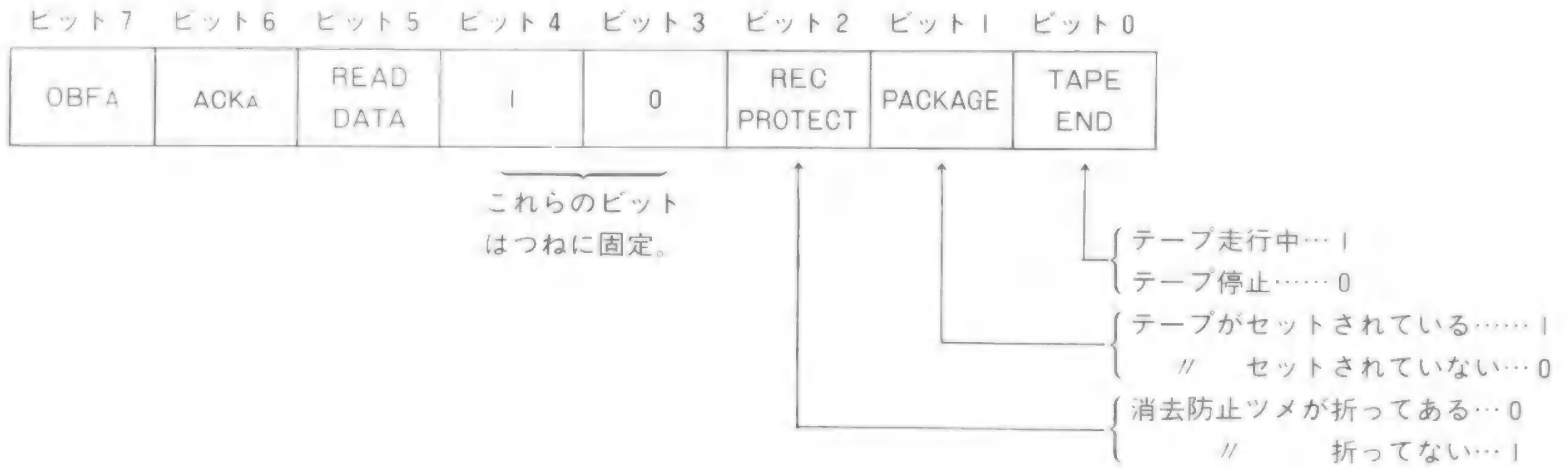
《サブ側　8255　ポートB》

ビット7	ビット6	ビット5	ビット4	ビット3	ビット2	ビット1	ビット0
OBFA	ACKA	READ DATA	1	0	REC PROTECT	PACKAGE	TAPE END

ビット4・ビット3：これらのビットはつねに固定。

ビット0（TAPE END）：テープ走行中…1／テープ停止……0

ビット1（PACKAGE）：テープがセットされている……1／〃　セットされていない…0

ビット2（REC PROTECT）：消去防止ツメが折ってある…0／〃　折ってない…1

HuBASICのIOCS内のシステムサブルーチン 0DF6H をコールすると、ポートBの情報がAレジスタに入るので、CMT (x) というBASICコマンドは、このビット x の位置を見ているのです。次に、サブ側8255のポートBを読むプログラムを掲げます。

リスト　　サブ側8255ポートBを調べる

```
100 REM ******************************
110 REM *                            *
120 REM *  ｻﾌﾞ 8255 ﾎﾟｰﾄB ｦ ｼﾗﾍﾞﾙ    *
130 REM *                            *
140 REM ******************************
150 '
160 CLEAR &HE000
170 '
180 WIDTH 40 : INIT : CLS
190 PRINT "ｻﾌﾞ 8255 port B"
200 PRINT : PRINT
210 '
220 REM *** ﾏｼﾝｺﾞ ｶｷｺﾐ ***
230 '
240 ADR=&HE000+65536! : RESTORE 350
250 '
260 READ MC$
270 IF MC$="END" THEN 420
280 MC=VAL("&H"+MC$)
290 POKE ADR,MC
300 ADR=ADR+1
310 GOTO 260
320 '
330 REM *** ﾏｼﾝｺﾞ ﾃﾞｰﾀ ***
340 '
350 DATA CD,F6,0D  : REM  CALL 0DF6H
360 DATA 32,10,E0  : REM  LD    (E010H),A
370 DATA C9        : REM  RET
380 DATA END,      : REM  data end mark
390 '
400 REM *** ﾒｲﾝ･ﾙｰﾌﾟ ***
410 '
420 CALL &HE000      : REM ﾎﾟｰﾄ B ｦ ﾖﾑ
430 A=PEEK(&HE010)
440 LOCATE 0,3
450 PRINT RIGHT$("0000000"+BIN$(A),8)
460 GOTO 420
```

プログラムを走らせると、ポートBのビットパターンが表示されます。カセットをEJECTしたり、テープを早送りしたり、消去防止ツメを折ってあるテープを入れてみたりして、ビットパターンの変化を調べて下さい。

ユーザーのマシン語プログラムで、テープの状態を調べる方法はもうおわかりですね(例：テープの走行チェック等)。

第9章　IPL ROM

9－1　IPLについて

X 1シリーズは、MZシリーズと同様に、シャープ伝統のクリーン設計で作られており、64 Kバイトのメインメモリーはすべて RAM となっています。しかし、これでは電源ONの時、メモリー内は空で、実行すべきプログラムがありません。

この状態から、システムを立ち上がらせる過程を**ブート・ストラップ（bootstrap）**とよびます。メインメモリーがすべて RAM で構成されているコンピュータでは、この過程は大変スリリングで、コンピュータが自分で自分を引っぱり上げるような動作をします。（ブートストラップの原理は文献〔20〕の第2章に劇的に（？）語られていますから、興味のある方は参照して下さい。）ブートストラップとは「靴ひも」の意味ですね。ユーモアを含んだ命名と思われますが、一説によると、底なし沼に落ちたほら吹き男爵が、自分の靴ひも（bootstrap）にぶら下がって、自分で自分を沼から持ち上げて脱出した話にちなんでいるそうです。

いずれにしても、ブートストラップ処理過程の目的は、最初に実行されるシステム・プログラムをメインメモリーにロードすることで、この意味から、**イニシャル・プログラム・ローダ（Initial Program Loader）**、略して、IPL ともよばれます。

〔注〕X 1 の HuBASIC コマンドに、IPL を起動させるための BOOT というのがありますが、これは上述のブートストラップからとった名称です。

9－2　X 1におけるIPL

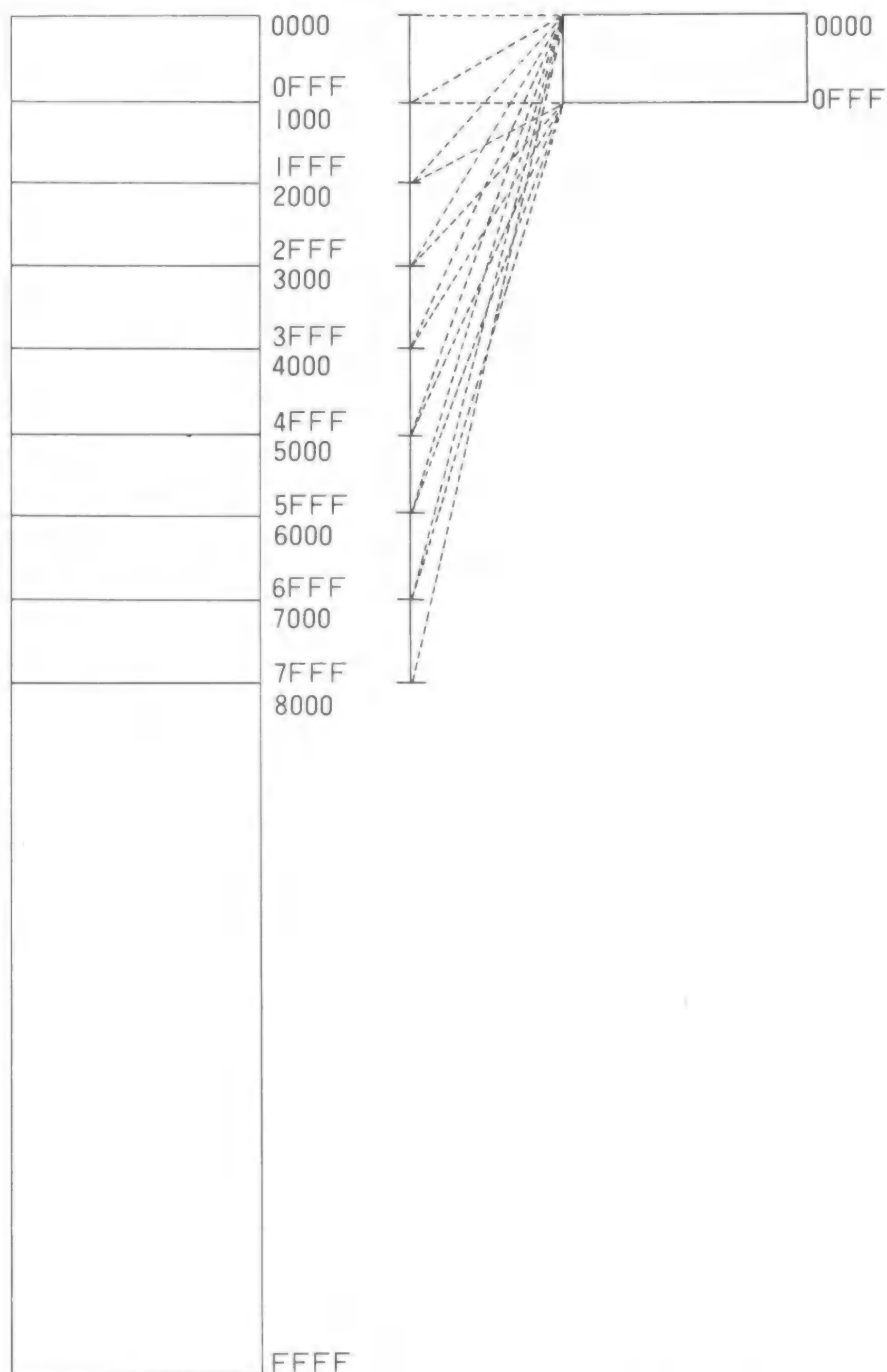
話をX 1に戻しましょう。X 1シリーズでは、IPL は4 Kバイト分の ROM に格納してあって、電源ON の時には、IPLROM が実行されます。

X 1の電源を入れると、

"IPL is under preparing"（IPL は準備中）
"IPL is looking for a program"（IPL はプログラム探索中）
"Make ready any device"（周辺装置を指定せよ）

などのメッセージがディスプレイ・テレビに出ますが、これは IPL が動作しているためです。カセットやフロッピーディスクにシステムプログラムが入っている時には、IPL がこれらをメインメモリーにロードしてくれる訳ですね。

ところで、読者の中には不思議に思われた方もおられるかもしれません。X 1シリーズでは、メモリー 64 K バイトはすべて RAM でした。 IPLROM はどこにあるのでしょうか？ 状況は次図のようになっています。

X 1におけるメインメモリーと IPLROM の関係

IPLROM は、0000 H～0 FFFH 番地に配置され、メインメモリーと番地が重なってしまいます。同一番地に複数にメモリーが配置されていても、どちらか一方しかアクセスできませんから、その時々にメモリーの選択をしなくてはいけません。これを**バンク切り替え**とよびます。(**バンク**というのは、同一番地に配置されている各メモリー・ブロックのこと)。

X 1の電源をONにすると、まず IPLROM の方が選択されます。面白いことに、この状態ではメモリーの0000 H～7 FFFH 番地がすべて、0000 H～0 FFF 番地に「折りたたまれて」しまいます。たとえば、7000 H～7 FFFH 番地は、0000 H～0 FFFH 番地と全く同等になります。

こうして、IPL 処理が実行されますが、ここでもまた疑問を持たれる方がおられるでしょう。「0000 H～0 FFFH 番地は ROM である。ROM は書き込めない。ではこの部分に入るシステムプログラムはどのようになるのか？」という疑問かと思われます。X 1においては、この問題を「IPL においては、読み出しで ROM が、書き込みで RAM が選択される」と設計することで解決してしまいます。このようにして、IPL プログラムにより、カセットテープやディスクから読み出されたシステムプログラムは、RAM に書き込まれる訳ですね。

9－3　バンク切り替え

IPLROM は、システムをメインメモリー（RAM）にロードし終えると、バンク切り替えによりメインメモリーから切り離されます。

このようなバンク切り替えは、I／O ポートに設けられているスイッチにより実行されます。次の2つのI／O アドレスが大切です。

《バンク切り替え関連I／Oポート》

I／Oアドレス	1D00H番地〔注〕
機　　能	この番地にデータ（何でもよい）を出力すると、0000H～7FFFH番地がIPLROMのバンクに切り替わる。

I／Oアドレス	1E00H番地〔注〕
機　　能	この番地にデータ（何でもよい）を出力すると、0000H～7FFFH番地がRAMのバンクに切り替わる。

〔注〕 上のいずれのI／Oアドレスも、有効なのはアドレス上位8ビットのみですから、アドレスの下位8ビットは何でも構いません。たとえば、I／Oアドレス1DFFH番地も、1D00H番地と同じ機能を持ちます。

たとえば、次のプログラムはいずれも IPLROM のバンクへの切り替えをします。

```
〔例1〕    LD    A, 1DH
           OUT   (00H), A        ;I／O (1D00H) ← 1DH
〔例2〕    LD    BC, 1D00H
           OUT   (C), A          ;I／O (1D00H) ← Aレジスタの内容
```

これを利用すると、IPLROM の内容をメインメモリー(RAM)に転送するプログラムを作ることができます。

《IPLROM内容の転送》

マシンコード	ニーモニック	コ メ ン ト
210000	LD HL, 0000H	元の先頭アドレス
1100E0	LD DE, 0E000H	転送先の先頭アドレス
010010	LD BC, 1000H	転送バイト数
3E1D	LD A, 1DH	
D300	OUT (00H), A	バンクを IPLROM に切り替え
EDB0	LDIR	ブロック転送
3E1E	LD A, 1EH	
D300	OUT (00H), A	バンクを RAM に切り替え
C30010	JP 1000H	Huモニター・ホット・スタートへジャンプ

このプログラムは「リロケータブル」ですから、転送先に使用する E 000 H～EFFFH 番地を避ければ、どこに配置しても構いませんが、IPLROM へのバンク切り替えでは、0000 H～7 FFFH 番地が ROM バンクに替わってしまいますから、この範囲に置くと、暴走したり、IPL が起動したりしますから避けて下さい。D 000 H 番地～のあたりに配置するのが適当でしょう。

まあともかく、上のプログラムをメモリーに書き込み、Hu モニターの G コマンドで実行すると、一瞬のうちに終了し、モニターのコマンド待ちに戻りますから、＊D E 000 EFFF でメモリーをダンプして下さい。このようにして、IPLROM の内容を見ることができる訳です。

《IPLROM の内容を見る》

```
Ok
MON
*MD000
:D000=210000
:D003=1100E0
:D006=010010
:D009=3E1D
:D00B=D300
:D00D=EDB0
:D00F=3E1E
:D011=D300
:D013=C30010
:D016=00
*GD000
*■

*DE000
:E000=C3 06 00 C3 00 07 F3 01 00 40 21 00 80 16 FF ED /テ..テ..ホ..@!._.テ▪
:E010=78 7E 15 20 FA 31 00 00 01 03 1A 3E 82 ED 79 0B /x‾. ﾀ1.....>_▪y.
:E020=ED 78 3E 72 ED 79 0B ED 78 CD 51 04 CD 2B 04 3E /▪x>r▪y.▪x\Q.\+.>
:E030=07 32 86 FF AF 32 87 FF CD 7E 04 3E E4 CD 2E 05 /.2■テｯ2■テ\‾.>◆\..
:E040=AF CD 2E 05 CD 0E 01 CD A1 01 01 00 0E CD 53 01 /ｯ\..\..\......\S.
:E050=28 42 CD C5 00 11 EC 0B CD 6F 01 21 00 FF 01 20 /(B\ﾅ...▪.\o.!.テ.
:E060=00 CD 67 05 18 03 C3 06 00 3A 00 FF FE 01 C2 E8 /.\g...テ..:.テ主.ｯ▪
:E070=00 CD 4F 05 2A 14 FF ED 4B 12 FF CD 64 05 3E 04 /.\O.*.テ▪K.テ\d.>.
:E080=CD 48 06 21 40 01 11 78 FF 01 06 00 ED B0 2A 16 /\H.!@..xテ...▪‾*.
:E090=FF C3 78 FF 01 00 0E CD 53 01 20 59 2E 00 D9 21 /テテxテ...\S. Y..ﾙ!
:E0A0=00 FF 01 20 00 CD 46 01 CD 4F 05 D9 2A 1D FF ED /.テ. .\F.\O.ﾙ*.テ▪
:E0B0=5B 1E FF D9 2A 14 FF ED 4B 12 FF CD 46 01 18 C3 /[.テﾙ*.テ▪K.テ\F..テ
:E0C0=CD 00 07 18 30 CD 52 06 E6 02 C0 AF CD 48 06 11 /\...0\R.◢.ﾀｯ\H..
:E0D0=08 0C 31 00 00 CD CB 03 11 65 0B CD CB 03 CD 0E /..1..\ﾋ..e.\ﾋ.\.
:E0E0=01 CD 52 06 E6 02 C2 52 00 18 F3 3E 04 CD 48 06 /.\R.◢.ﾂR..ホ>.\H.
:E0F0=11 20 0C 18 08 11 08 0C 18 03 11 F3 0B 31 00 00 /. ..........ホ.1..
*■
```

9－4 IPL ROMの内部解析

前節のようにして、IPLROM の内容を RAM 上に転送して、内部解析した結果を以下に参考資料として掲げます。

HuBASIC のコマンドで、IPL に関連するものが2つあります。BOOT と ASK がそれですね。BOOT コマンドを実行すると、IPL のバンク切り替え後、0000 H 番地へジャンプするので、IPL が起動します。これに対し、ASK コマンドを実行すると、バンク切り替え後、0003 H 番地がコールされて、TV タイマーが起動します。

IPLROM は、4 K バイトと短いし、基本的な入出力ルーチンが納められているので、解析してみると、X 1のハードウェアに対する理解が深まるでしょう。

アドレス	機　　能
0000 0003	IPL起動　　(JP 0006H) TVタイマー起動(JP 0700H)
0006 0038 0044 0047 004A	システム初期化 IPLの準備(サブCPUの動作チェック) キー入力待ち(電源ONの最初は実行されない) フロッピーディスクからプログラムを探す ROMカードの装着チェック(装着時は、0094Hへジャンプ)
0052 0066 0069 007E 0083	デバイスとしてCMT(カセット)を指定した時の処理 ノンマスカブル・インタラプト(JP 0006H) CMT処理(続き) カセット・テープを巻き戻し IPLの切り離し
0094	デバイスとしてROMを指定した時の処理
00C0	デバイスとしてTIMERを指定した時の処理
00C5	デバイス選択処理
00EB 00F5 00FA	File mode error SHIFT + BREAK 処理 Program load error
0103	Push … Key … と表示して、キー入力待ち
010E	キー入力処理(キースキャン)
0140	RAM バンクにして、HL 番地へジャンプ
016F	IPL is looking for a program from …… と表示(DE にデバイス・メッセージ・ポイントを入れて、コール)
017A 0188 01A1 01CF 01E9	デバイスとして、フロッピー・ディスクを指定した時の処理 ドライブ番号の指定 〝IPL is looking for a program from FD〟と表示 〝IPL is loading …〟と表示 IPL 切り離し

アドレス	機　　　　能
01ED	ディスクよりのインフォメーション・ブロック・ロード
0202	ファイル属性01Hと、拡張子"Sys"のチェック
021A	ディスクからのロード
0355	カーソル点滅して、キー入力待ち
038A	キー入力サブ
03CB	メッセージ表示(DEをポインタ、00Hをエンドマークとする)
03D4	キャラクタ"X"表示
03D9	1文字表示
03E3	カーソル制御
03FB	VRAMアドレス計算
0410	コントロール・コード実行
042B	画面クリア
044A	改行処理
0451	CRTC初期化
0465	パレット・プライオリティー初期化(すべて00H)
0471	PSG初期化(音声ミュート)
047E	サブCPUの動作チェック(IPL is under preparingと表示)
04C5	動作不良エラー("Please turn on the power SW slowly again !"と表示)
0523	サブCPUからのデータ入力
052E	サブCPUへのデータ出力
053B	サブCPUからの受信準備待ち
0545	サブCPUへの送信準備待ち
054F	"IPL is loading ファイル名"を表示
0564	カセットからのデータ・ブロック・ロード
0567	カセットからのインフォメーション・ブロック・ロード
0648	カセット制御
0652	テープ状態を得る
065A	サブCPUより「年・月・曜・日」を得る
066C	サブCPUより「時・分・秒」を得る
0673	年・月・日・曜・時・分・秒の表示
06A3	"／"を表示
06A8	(HL)を16進表示
06AD	(HL)を16進表示し":"をつける。
06B5	月、日、曜の表示
06E5	A＝00Hなら"X"を表示
06E9	Aレジスタの内容を16進表示

アドレス	機　　能
0700	TVタイマー制御
077C	プロンプト・メッセージ表示
0785	キースキャン
07DF	"CH"表示
07E8	"OFF"表示
07F1	"ON CH"表示
07FA	→ 処理
0810	← 処理
082C	↓ 処理
083B	↑ 処理
0841	CLR 処理
0877	"DATA Error!!"表示
089B	タイマー時、メッセージ表示
08B7	時計とタイマーの場合分け
08BE	タイマーの設定、変更処理
0953	時計の設定、変更処理
09CC	サブCPUへの3バイト出力(タイマー用)
09E0	タイマー用入力処理
0AA0	サブCPUからの6バイト入力(タイマー用)
0AAB	TIMER n の表示(nは1～7)
0B2D	メッセージを消す
0B33	BCD → 数値変換(月の数字用)
0B3E	CRTC出力データ　(14バイト)
0B4C	カーソル制御データ(時計用x座標)
0B54	〃　　　(タイマー用x座標)
0B5D	〃　　　(y座標)
0B65	メッセージ・データ(Push …… Key)
0BA0	メッセージ・データ(Drive No ? ……)
0BB7	メッセージ・データ(IPL is loading)
0BC8	メッセージ・データ(IPL is looking ……)
0BEC	メッセージ・データ(CMT)
0BF0	メッセージ・データ(FD)
0BF3	メッセージ・データ(Program load error)
0C08	メッセージ・データ(Make ready ……)
0C20	メッセージ・データ(File mode error)
0C35	メッセージ・データ(IPL is under preparing)
0C4D	メッセージ・データ(Please turn ……)
0C94	メッセージ・データ(TV Timer control 倍文字用)
0CB5	メッセージ・データ(DATA error !!)
0CDC	メッセージ・アドレス・テーブル
0D01	プロンプト・メッセージ・データ(Year ……)
0D28	プロンプト・メッセージ・データ(Month ……)
0D4F	プロンプト・メッセージ・データ(Day ……)
0D76	プロンプト・メッセージ・データ(SUN MON ……)

アドレス	機能
0D9D	プロンプト・メッセージ・データ(Hour……)
0DC4	プロンプト・メッセージ・データ(Minute……)
0DEB	プロンプト・メッセージ・データ(Second……)
0E12	プロンプト・メッセージ・データ(TV Power……)
0E39	プロンプト・メッセージ・データ(TV Channel……)
0E60	プロンプト・メッセージ・データ(消去用)
0E87	曜日文字列(4×8バイト)
0EA9	文字列(ON CH)
0EAD	文字列(TIMER)
0EB3	文字列(XX/XX XXX……)
0ECF	文字列([ESC]=Exit, [CLR]=……)
0EF1～0EF2	文字列(# D)

IPL用ワーク・エリア

アドレス	機能
FF00	ファイル属性
FF01～FF11	ファイル名（17字分。拡張子、パスワード含む）
FF12	ファイル長（2バイト）
FF14	ロード先頭アドレス（2バイト）
FF16	実行先頭アドレス（2バイト）
FF18	タイマー・データ（6バイト）
FF1E	（ディスク時）先頭クラスタ
FF20	サブCPUバッファ（6バイト。タイマー用）
FF26	曜日文字列バッファ（3バイト）
FF80	カーソルx座標
FF81	カーソルy座標
FF82	注目タイマー番号
FF83	チェックサムバッファ（2バイト）
FF85	キー入力ワーク
FF86	テキスト・アトリビュート
FF87	ドライブ番号（0～3）
FF88	SP退避エリア（2バイト）
FF8A	エラー時、ジャンプ先（2バイト）
FF8C～	汎用ワーク・エリア
～FFFF	IPL時スタック・エリア

第10章　漢字ROM

10－1　漢字コード

HuBASICでは、KANJI$という関数があって、漢字をグラフィック画面に表示できるようになっています。漢字ROMボード（CZ-8KR）を装着すると、KANJI$が使えるようになります。

さて、KANJI$関数では、**図形文字用符号コード一覧表**に示されている**区**と**点**を指定することで、そのコード（以下、**区点コード**とよぶことにします）に対応する漢字パターンを漢字ROMから読みとります。たとえば、亜という漢字の区点コードは、1601(16区，01点）ですから、KANJI$(1601)とします。

漢字コードには、上記の区点コードの他に、**JIS漢字コード**とよばれる、JIS規格で決まっているコードがあります。漢字ROMに納められている「JIS第1水準」の漢字・非漢字に対して、区点コードとJIS漢字コードとは次の対応があります。

区点コード		JIS漢字コード	
区の番号	20Hを加える→	上位1バイト	これらを合わせて16進4桁で指定する
点の番号	20Hを加える→	下位1バイト	

従って、亜のJIS漢字コードは、3021Hとなります。

漢字ROMのアクセスには、これら2つの漢字コードが組み合わさって用いられます(JIS漢字コードの上位バイトと、点番号)。

10－2　漢字ROMのI／Oアドレス

漢字ROMボードCZ-8KRは、拡張I／Oポートに装着され、アクセスするI／Oアドレスはボード上の回路により、次のように決められています。

《漢字ROM関係Ｉ／Ｏポート》

I／Oアドレス	入出力	機　　　　能
0E82H	出　力	00H 出力…… ROM からのデータ読み出し終了 01H 出力…… ROM からのデータ読み出し開始
0E81H 0E80H	入出力	I／O(0E81H)← 00H I／O(0E80H)← JIS 漢字コード上位バイト } のとき データ読み出し時には、 I／O(0E81H)からは 00H, I／O(0E80H)からは ROM 内アドレス上位バイト が得られる。
		I／O(0E81H)← ROM アドレス上位バイト(≠00H) I／O(0E80H)← ROM 内アドレス下位バイト } のとき データ読み出し時には、 I／O(0E81H)からは、右部分フォントパターン16バイト分 I／O(0E80H)からは、左部分フォントパターン16バイト分 が得られる。

これらのＩ／Ｏポートの使い方の例として、区点コードを漢字 ROM 内のデータ格納アドレスに変換する BASIC プログラムを紹介します。これより、 亜 のフォント・パターンは ROM 内の 4010 H～401 FH に格納されていることがわかります。

リスト　　　漢字コード→ ROMアドレス　（BASIC）

```
100 REM ********************************
110 REM *                              *
120 REM *  ｶﾝｼﾞ ｺｰﾄﾞ --> ROM ｱﾄﾞﾚｽ  *
130 REM *                              *
140 REM ********************************
150 '
160 REM -- ﾆｭｳﾘｮｸ --
170 '
180 PRINT
190 INPUT "ｸﾃﾝ ｺｰﾄﾞ   = ";CODE
200 KU  =(CODE ¥ 100)
210 TEN =(CODE MOD 100)
220 JISH=KU+32
230 '
240 REM -- ｱﾄﾞﾚｽ ﾃﾞｰﾀ ﾉ ﾖﾐﾀﾞｼ --
250 '
260 OUT &HE80,JISH
270 OUT &HE81,0
280 '
290 OUT &HE82,1      :REM ﾖﾐﾀﾞｼ ｶｲｼ
300 ADH=INP(&HE80)
310 ADL=INP(&HE81)
320 OUT &HE82,0      :REM ﾖﾐﾀﾞｼ ｼｭｳﾘｮｳ
330 '
340 REM -- ROM ｱﾄﾞﾚｽ ﾉ ｹｲｻﾝ --
350 '
360 ROMADR=ADH*256+TEN*16
370 PRINT "ROM ｱﾄﾞﾚｽ =    ";HEX$(ROMADR)+"H"
380 '
390 GOTO 180
```

```
RUN

ｸﾃﾝ ｺｰﾄﾞ   = ? 1601
ROM ｱﾄﾞﾚｽ =    4010H

ｸﾃﾝ ｺｰﾄﾞ   = ? 1602
ROM ｱﾄﾞﾚｽ =    4020H

ｸﾃﾝ ｺｰﾄﾞ   = ? ■
```

10－3　漢字のフォント・パターンを読み出す

では、いよいよ漢字 ROM から漢字フォントのパターンを読み出してみましょう。

前節で得られた ROM 内データのアドレスを、I／O ポートの 0 E 80 H 番地（下位アドレス)、0 E 81 H 番地(上位アドレス)に出力してから、読み出しにかかります。フォント・データは 16 バイト分得られ、I／O ポート 0 E 80 H からは左半分、0 E 81 H からは右半分のパターンが読み出されます。

前節のプログラムに続けて、フォント読み出し部分を加えたものが次に掲げるプログラムです。区点コード 1601 を入力すると、見事に 亜 のフォント・パターンが出力される様を味わって下さい。

リスト　　　漢字フォントを読み出す（BASIC）

```
100 REM ********************************
110 REM *                              *
120 REM *      ｶﾝｼﾞ ﾌｫﾝﾄ ｦ ﾖﾐﾀﾞｽ        *
130 REM *                              *
140 REM ********************************
150 '
160 REM -- ﾆｭｳﾘｮｸ --
170 '
180 PRINT
190 INPUT "ｸﾃﾝ ｺｰﾄﾞ   = ";CODE
200 KU  =(CODE ¥ 100)
210 TEN =(CODE MOD 100)
220 JISH=KU+32
230 '
240 REM -- ｱﾄﾞﾚｽ ﾃﾞｰﾀ ﾉ ﾖﾐﾀﾞｼ --
250 '
260 OUT &HE80,JISH
270 OUT &HE81,0
280 '
290 OUT &HE82,1      :REM ﾖﾐﾀﾞｼ ｶｲｼ
300 ADH=INP(&HE80)
310 ADL=INP(&HE81)
320 OUT &HE82,0      :REM ﾖﾐﾀﾞｼ ｼｭｳﾘｮｳ
330 '
340 REM -- ROM ｱﾄﾞﾚｽ ﾉ ｹｲｻﾝ --
350 '
360 ROMADR=ADH*256+TEN*16
370 PRINT "ROM ｱﾄﾞﾚｽ =    ";HEX$(ROMADR)+"H"
380 '
390 REM -- ﾌｫﾝﾄ ﾉ ﾖﾐﾀﾞｼ --
400 '
410 DIM FONT(16,2)
420 '
430 OUT &HE80,(ROMADR MOD 256)   :REM ｱﾄﾞﾚｽ ｼﾃｲ
440 OUT &HE81,(ROMADR ¥ 256)
450 '
460 FOR I=1 TO 16
470   OUT &HE82,1             :REM ﾖﾐﾀﾞｼ ｶｲｼ
480   FONT(I,1)=INP(&HE80)  :REM ﾋﾀﾞﾘ ﾊﾟﾀｰﾝ
490   FONT(I,2)=INP(&HE81)  :REM ﾐｷﾞ  ﾊﾟﾀｰﾝ
500   OUT &HE82,0             :REM ﾖﾐﾀﾞｼ ｼｭｳﾘｮｳ
510 NEXT I
520 '
530 REM -- ﾌｫﾝﾄ ﾉ ﾋｮｳｼﾞ --
540 '
550 PRINT
560 PRINT "ﾌｫﾝﾄ ﾊﾟﾀｰﾝ"
570 PRINT
580 FOR I=1 TO 16
590   L$=RIGHT$("0000000"+BIN$(FONT(I,1)),8)
600   R$=RIGHT$("0000000"+BIN$(FONT(I,2)),8)
610   PRINT L$+R$
620 NEXT I
630 '
640 PRINT
650 END
```

```
RUN

ｸﾃﾝ ｺｰﾄﾞ   = ? 1601
ROM ｱﾄﾞﾚｽ =    4010H

ﾌｫﾝﾄ ﾊﾟﾀｰﾝ

0000000000000000
0011111111111110
0000001000100000
0000001000100000
0000001000100000
0011111111111110
0010001000100010
0010001000100010
0010001000100010
0010001000100010
0010001000100010
0011111111111110
0000001000100000
0000001000100000
0000001000100000
0111111111111111
```

同様のプログラムをマシン語で組んだものが以下のリストです。一応、HL レジスタに亜 の区コード、DE レジスタに点コードをセットしておきましたが、他の漢字のフォントを読みたければ、330 行と 340 行に好きなコードを書いて下さい。

32 バイト分のデータは、E 060 H 番地以降に格納されます。E 060 H～E 06 FH が左半分のデータ、E 070 H～E 07 FH が右半分のデータに対応しています。BASIC 版とプログラムの基本構造は同じですが、雰囲気は随分違いますね。

リスト　　漢字フォント（MAS）

```
100 REM ************************
110 REM     ｶﾝｼﾞ ﾌｫﾝﾄ (MAS)
120 REM ************************
130 '
140 CLEAR &HE000
150 GOSUB "ｶｷｺﾐ"
160 CALL &HE000
170 PRINT "ﾃﾞｰﾀ : E060H ｶﾗ "
180 END
190 '
200 LABEL "ｶｷｺﾐ" :REM -------------------------------------------------
210 '
220 ADR=&HE000
230 READ MC$
240 IF MC$="END" THEN RETURN
250 POKE ADR,VAL("&H"+MC$)
260 ADR=ADR+1
270 GOTO 230
280 '
290 REM --- ﾏｼﾝｺﾞ ﾙｰﾁﾝ ---------------------------------------------
300 '
310                     :REM          ORG 0E000H
320                     :REM ;
330 DATA 21,10,00       :REM CODE:    LD   HL,16        ;ｸ
340 DATA 11,01,00       :REM          LD   DE,01        ;ﾃﾝ
350                     :REM ;
360 DATA 01,80,0E       :REM ADCAL:   LD   BC,0E80H
370 DATA 7D             :REM          LD   A,L
380 DATA C6,20          :REM          ADD  A,32         ;A=JIS code high
390 DATA ED,79          :REM          OUT  (C),A
400 DATA 0C             :REM          INC  C            ;BC=0E81H
410 DATA AF             :REM          XOR  A            ;A=0
420 DATA ED,79          :REM          OUT  (C),A
430 DATA 0C             :REM          INC  C            ;BC=0E82H
440 DATA 3C             :REM          INC  A            ;A=1
450 DATA ED,79          :REM          OUT  (C),A        ;read start
460 DATA 0D             :REM          DEC  C
470 DATA 0D             :REM          DEC  C            ;BC=0E80H
480 DATA ED,50          :REM          IN   D,(C)
490 DATA 0C             :REM          INC  C            ;BC=0E81H
500 DATA ED,78          :REM          IN   A,(C)
510 DATA AF             :REM          XOR  A            ;A=0
520 DATA 0C             :REM          INC  C            ;BC=0E82H
530 DATA ED,79          :REM          OUT  (C),A        ;read end
540 DATA 0D             :REM          DEC  C
550 DATA 0D             :REM          DEC  C            ;BC=0E80H
560 DATA 6B             :REM          LD   L,E
570 DATA 29             :REM          ADD  HL,HL
580 DATA 29             :REM          ADD  HL,HL
590 DATA 29             :REM          ADD  HL,HL
600 DATA 29             :REM          ADD  HL,HL        ;HL=ﾃﾝ*16
610 DATA 5F             :REM          LD   E,A          ;E=0
620 DATA 19             :REM          ADD  HL,DE        ;HL=ROM ｱﾄﾞﾚｽ
630 DATA ED,69          :REM FONTRD:  OUT  (C),L        ;adrs low
640 DATA 0C             :REM          INC  C            ;BC=0E81H
650 DATA ED,61          :REM          OUT  (C),H        ;adrs high
660 DATA DD,21,60,E0    :REM          LD   IX,0E060H    ;IX=data store adrs
670 DATA 0C             :REM          INC  C            ;BC=0E82H
680 DATA 16,10          :REM          LD   D,16         ;counter (16 bytes)
690 DATA 3E,01          :REM LOOP:    LD   A,1
700 DATA ED,79          :REM          OUT  (C),A        ;read start
710 DATA 0D             :REM          DEC  C
720 DATA 0D             :REM          DEC  C            ;BC=0E80H
730 DATA ED,78          :REM          IN   A,(C)        ;font of left
740 DATA DD,77,00       :REM          LD   (IX+0),A     ;store
750 DATA 0C             :REM          INC  C            ;BC=0E81H
760 DATA ED,78          :REM          IN   A,(C)        ;font of right
770 DATA DD,77,10       :REM          LD   (IX+16),A    ;store
```

```
780 DATA 0C           :REM        INC C       ;BC=0E82H
790 DATA AF           :REM        XOR A       ;A=0
800 DATA ED,79        :REM        OUT (C),A   ;read end
810 DATA DD,23        :REM        INC IX      ;data pointer inc
820 DATA 15           :REM        DEC D       ;counter dec
830 DATA 20,E6        :REM        JR  NZ,LOOP
840 DATA C9           :REM EXIT:  RET
850                   :REM ;
860 DATA END,         :REM end mark
870 '
880 REM -------------------------------------------------------------
```

第11章　フロッピー・ディスク

11－1　はじめに

X 1シリーズにおいては、2種類のフロッピー・ディスクがサポートされています。

ミニフロッピーディスク （5.25インチ）	ミニフロッピーディスクドライブ CZ-801Fにて使用可能。
コンパクトフロッピーディスク （3インチ）	XIDに標準装備。

本章では、一応 CZ-801 F のドライブ装置と 5.25 インチディスクを想定して話を進めますが、コンパクトフロッピーディスクでも本章の内容はそのまま通用します。

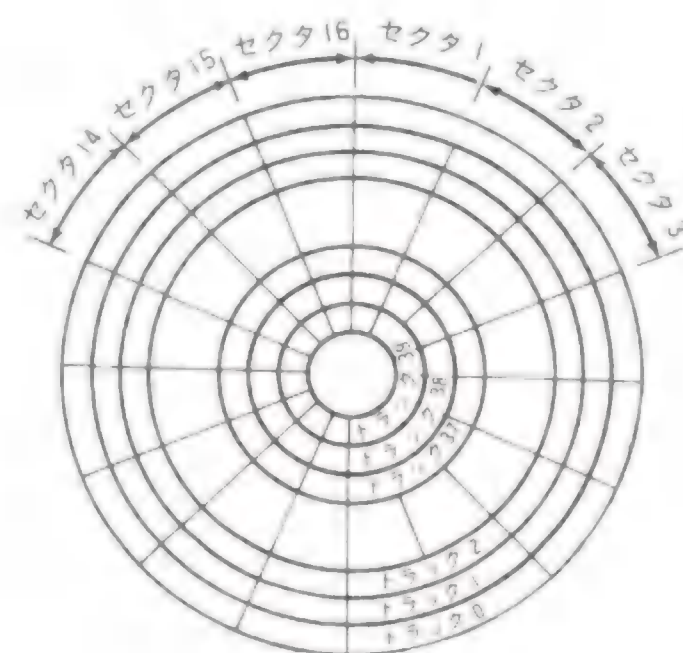
〔注〕 当然のことながら、本章の内容は、テープ BASIC では実行できませんから、登場するシステム・サブルーチンのアドレスは、すべてディスク BASIC のものです。

11－2　セクタ、トラック、ヘッド

フロッピーディスクには、目には見えませんが半径の異なる多数の同心円が書かれていて、その1つ1つは**トラック**（track）とよばれます。各トラックを区別するために、トラックには番号がつけられていて、外側から内側へ向って第0トラック～第39トラックとよばれます。トラックは計40個あります。

各トラックは円周状をしていて、これを中心から放射状に16個に分割して、その各々を**セクタ**（sector）とよびます。フロッピーディスクとのデータのやりとりは、通常セクタを単位として行なわれます。1つのトラック上のセクタには1～16の番号がつけられています。

以上の話をまとめると、次の図になります。

トラックとセクタ

X 1シリーズでは、画面倍密度のフロッピーディスクが使われるので、さらに表の面と裏の面が区別されます。これらは、**サイド0**、**サイド1**あるいは、各面に押しつけられる2個のヘッドにより、**ヘッド0**、**ヘッド1**とよぶことにします。

X 1シリーズの場合、1セクタには、256バイトのデータを記録できます。1トラックには、これらが16セクタあり、片方の面には40トラックありますから、両面全体で記録できるデータのバイト数は、

$$256\times16\times40\times2=327680\text{（バイト）}$$
$$=320\text{ K（バイト）〔注〕}$$

となります。

〔注〕1 Kバイト＝1024バイト。

11－3　物理アドレスと論理アドレス

前節で述べたように、データを記録する際の単位となるセクタの位置は、トラック番号、ヘッド番号、セクタ番号を指定すれば決まります。これを、そのセクタの**物理アドレス**（physical address）とよびます。

一方、X 1シリーズのディスクBASIC(CZ-8 FB 01)では、各セクタに連続した通し番号（**レコード番号**）をつけて管理しています。これをセクタの**論理アドレス**（logical address)とよびます。ディスクBASICでは、さらに16セクタを1かたまりと考えて、**クラスタ**（cluster）という単位で管理します。

物理アドレスと論理アドレスの関係は次の通りです。

物理アドレスと論理アドレス

論理アドレス	物理アドレス		
レコード番号	トラック番号	ヘッド番号	セクタ番号
0000H	0	0	1
0001H	0	0	2
0002H	0	0	3
⋮	⋮	⋮	⋮
000EH	0	0	15
000FH	0	0	16
0010H	0	1	1
0011H	0	1	2
0012H	0	1	3
⋮	⋮	⋮	⋮
001FH	0	1	16
0020H	1	0	1
0021H	1	0	2
0022H	1	0	3
⋮	⋮	⋮	⋮

論理アドレス	物理アドレス		
レコード番号	トラック番号	ヘッド番号	セクタ番号
⋮	⋮	⋮	⋮
002FH	1	0	16
0030H	1	1	1
0031H	1	1	2
⋮	⋮	⋮	⋮
04E0H	39	0	1
04E1H	39	0	2
⋮	⋮	⋮	⋮
04EEH	39	0	15
04EFH	39	0	16
04F0H	39	1	1
04F1H	39	1	2
⋮	⋮	⋮	⋮
04FEH	39	1	15
04FFH	39	1	16

レコード番号を 16 ビットで指定すると、そのビットの意味は次のようになっています。

《レコード番号のビットの意味》

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
0	0	0	0	0											

ビット10～5：トラック番号　ビット4：ヘッド番号　ビット3～0：セクタ番号-1

11－4　ディスクとの入出力ルーチン

以上を理解すると、ディスク BASIC 内のシステム・サブルーチンを使ってディスクへのデータの書き込みと、ディスクからのデータの読み出しをすることができます。

《ディスク関係システムサブルーチン》

名　　称	ディスクからのリード・データ
アドレス	75D3H
入力条件	DE レジスタ＝読み出し先頭セクタのレコード番号 A レジスタ　＝読み出すセクタの個数 HL レジスタ＝データ格納先頭アドレス
機　　能	ディスクの指定セクタより、指定個数だけデータを読み出し、メモリーに格納する。

名　　称	ディスクへのライト・データ
アドレス	765AH
入力条件	DE レジスタ＝書き込み先頭セクタのレコード番号 A レジスタ　＝書き込むセクタの個数 HL レジスタ＝出力データの格納先頭アドレス
機　　能	メモリー上に用意されたデータを、ディスクの指定範囲のセクタへ連続的に書き込む。

では早速実験をしてみましょう。まずディスクからのデータ読み出しの方から行ないます。

《レコード番号16のセクタを読む》

```
CLEAR &HE000
Ok
MON
*ME000
:E000=3E01
:E002=111000
:E005=2100F0
:E008=CDD375
:E00B=C9
:E00C=00
*GE000
*■
```

このプログラムでは、DE＝16, A＝1, HL＝F 000 H と設定して、レコード番号 16(16 進で 0010 H) のセクタを読み出し、その内容をメモリー上の F 000 H 番地から格納します。

当然のことながら、ディスク装置の電源を ON にし、ドライブ 0 にディスクを挿入した後に、実行して下さい。ディスク装置の動きを示すライトが点灯し、しばらくして動作終了となったら、F 000 H 以降をダンプして下さい。

《読み出し結果》

```
*DF000
:F000=11 42 41 53 49 43 20 43 /.BASIC C
:F008=5A 38 46 42 30 31 53 79 /Z8FB01Sy
:F010=73 20 00 A8 00 00 00 00 /s .ｨ.....
:F018=82 C1 27 17 58 00 02 00 /▂ﾁ'.X...
:F020=02 53 74 61 72 74 20 75 /.Start u
:F028=70 20 20 20 20 20 42 61 /p     Ba
:F030=73 20 05 01 00 00 00 00 /s ......
:F038=82 C1 27 18 15 00 0D 00 /▂ﾁ'.....
:F040=42 55 74 69 6C 69 74 79 /BUtility
:F048=20 20 20 20 20 20 20 20 /
:F050=20 20 CD 12 00 00 00 00 /  ﾍ.....
:F058=82 C1 27 18 25 00 0E 00 /▂ﾁ'.%...
:F060=41 55 74 69 6C 69 74 79 /AUtility
:F068=20 20 20 20 20 20 4F 62 /      Ob
:F070=6A 20 00 0A 00 F5 00 00 /j ...火..
:F078=82 C1 27 18 34 00 10 00 /▂ﾁ'.4...
*■
```

上の例は、システム・ディスク (BASIC の入っているディスク) に対して、レコード番号 16 のセクタを読んだものです。ASCII ダンプの部分を見ると、どこかで見た文字列がありますね。システムプログラムと、ユーティリティプログラムのファイル名らしいですね。

ディスクBASICでは、レコード番号16～31の16個の連続したセクタ（第1クラスタ）に、格納されているファイル名等のデータ（カセット・テープでいうと「インフォメーション・ブロック」に相当するもの）を集中的に記録しています。この部分を**ディレクトリ・テーブル**あるいは単に**ディレクトリ**とよびます。(directoryを辞書で引くと、「住所氏名録」などと出ています。）　今ダンプしたのは、システムディスクのディレクトリの最初の部分でした。ディレクトリについては次節で詳しく扱います。

実験を続けます。今度はディスクへの書き込みです。これは危険（？）な実験――ディスクのデータを壊す恐れがある！――なので、壊れてもいい使用済ディスクか、あるいは、ユーティリティ・プログラムで「イニシャライズ」したディスクを用意し、ドライブ0に挿入して下さい。準備はよろしいですか？

では、このディスクのレコード番号04 FFHのセクタに次の無意味なデータを書き込んでみましょう。まず、サンプル・データをF000H～F0FFH番地に作成し、それから書き込みルーチンを実行します。

《サンプル・データ》

```
LIST
10 FOR I=0 TO 255
20    POKE &HF000+I,I
30 NEXT I
40 END
Ok
RUN
Ok
■
```

```
MON
*DF000
:F000=00  01  02  03  04  05  06  07  /........
:F008=08  09  0A  0B  0C  0D  0E  0F  /........
:F010=10  11  12  13  14  15  16  17  /........
:F018=18  19  1A  1B  1C  1D  1E  1F  /........
:F020=20  21  22  23  24  25  26  27  / !"#$%&'
:F028=28  29  2A  2B  2C  2D  2E  2F  /()*+,-./
:F030=30  31  32  33  34  35  36  37  /01234567
:F038=38  39  3A  3B  3C  3D  3E  3F  /89:;<=>?
:F040=40  41  42  43  44  45  46  47  /@ABCDEFG
:F048=48  49  4A  4B  4C  4D  4E  4F  /HIJKLMNO
:F050=50  51  52  53  54  55  56  57  /PQRSTUVW
:F058=58  59  5A  5B  5C  5D  5E  5F  /XYZ[¥]^_
:F060=60  61  62  63  64  65  66  67  /`abcdefg
:F068=68  69  6A  6B  6C  6D  6E  6F  /hijklmno
:F070=70  71  72  73  74  75  76  77  /pqrstuvw
:F078=78  79  7A  7B  7C  7D  7E  7F  /xyz{|}￣π
*■
```

```
*DF080
:F080=80  81  82  83  84  85  86  87  /
:F088=88  89  8A  8B  8C  8D  8E  8F  /
:F090=90  91  92  93  94  95  96  97  /
:F098=98  99  9A  9B  9C  9D  9E  9F  /
:F0A0=A0  A1  A2  A3  A4  A5  A6  A7  / 。「」、・ヲァ
:F0A8=A8  A9  AA  AB  AC  AD  AE  AF  /ィゥェォャュョッ
:F0B0=B0  B1  B2  B3  B4  B5  B6  B7  /ーアイウエオカキ
:F0B8=B8  B9  BA  BB  BC  BD  BE  BF  /クケコサシスセソ
:F0C0=C0  C1  C2  C3  C4  C5  C6  C7  /タチツテトナニヌ
:F0C8=C8  C9  CA  CB  CC  CD  CE  CF  /ネノハヒフヘホマ
:F0D0=D0  D1  D2  D3  D4  D5  D6  D7  /ミムメモヤユヨラ
:F0D8=D8  D9  DA  DB  DC  DD  DE  DF  /リルレロワン゛゜
:F0E0=E0  E1  E2  E3  E4  E5  E6  E7  /●○♠♥♦♣
:F0E8=E8  E9  EA  EB  EC  ED  EE  EF  /
:F0F0=F0  F1  F2  F3  F4  F5  F6  F7  / 土金木水火月日
:F0F8=F8  F9  FA  FB  FC  FD  FE  FF  /時分秒年円人生〒
*■
```

《データの書き込み》

```
*ME000
:E000=3E01
:E002=11FF04
:E005=2100F0
:E008=CD5A76
:E00B=C9
:E00C=00
*GE000
*■
```

予想通りデータが書き込まれているか否かのチェックは、レコード番号04 FFHのセクタを読んでみればよい訳です。確認できれば成功です。

このようにセクタ単位の読み出し、書き込みができますから、セクタのデータ変更は簡単にできますね。すなわち、

リード・データ → メモリー上で変更 → ライト・データ

とすればよい訳です。

11－5　ディレクトリ・テーブル

ディレクトリ・テーブルは、レコード番号16～31の位置にあって、ファイルの名前、属性、SAVE年月日時分、ファイル先頭クラスタ番号等を管理している領域です。1ファイルあたり32バイトから構成されていて、各バイトの意味は次のようになっています。

バイトの位置	内容
第0バイト	ファイル属性を表わす。 ただし00Hは、KILLされたファイル、 FFHは、ディレクトリテーブル使用域の終りを意味する。 その他のコードは、各ビットに次の意味がある。 ビット0……1のときBinファイル(マシン語ファイル) ビット1……1のときBasファイル(BASICのテキスト・ファイル) ビット2……1のときAscファイル(ASCIIセーブされたファイル) ビット4……1のときFILES表示あり、0のとき表示しない。 ビット5……1のときリードアフターライトON、0のときOFF　[注1] ビット6……1のとき書き込み禁止、0のとき書き込みOK　[注1] ビット3とビット7は予備。
第1バイト～第13バイト	ファイル名（13文字）
第14バイト～第16バイト	ユーザー指定ファイル拡張子（3文字）　[注2]
第17バイト	パスワード（20Hは無指定）　[注3]
第18、19バイト	ファイル・サイズ（バイト数）　※Bas, Binのみ有効
第20、21バイト	ファイルのメモリー格納先頭アドレス　※Binのみ有効
第22、23バイト	ファイルのメモリー実行アドレス　※Binのみ有効
第24バイト～第28バイト	SAVEした年、月、曜日、日、時、分を示す5バイト・データ　[注4]
第29バイト	00H（未使用）
第30バイト	ファイルが格納されている先頭クラスタ番号　[注5]
第31バイト	00H（未使用）

〔注1〕ディスクBASICのSETコマンドにより指定されます。
〔注2〕たとえばファイル名として"TEST.（1）"とするとピリオド以下の3文字"（1）"がファイル拡張子となります。同一のファイル名を区別するため用いられます。普通の場合は特につける必要はないが、IPLより起動するファイルは、"Sys"と指定しなければなりません。
〔注3〕前節の方法で、ディレクトリの変更ができますから、このバイトにパスワードを書くことは可能ですが、詳細はまだ不明です。
〔注4〕サブCPUの章、カセットの章でのタイマー・データの形式と全く同じです。
〔注5〕1クラスタ＝16レコードですから、ファイルが格納されている先頭セクタのレコード番号は、この値を16倍したものです。（16進表示では、末尾に0を1つつければよい。）

たとえば、前節でダンプしてみたディレクトリの先頭ファイルについて、各バイトの意味を考えると次のようになります。

ディレクトリの見方

コード	意味
11	00010001 { Binファイル FILES表示あり リード・アフタ・ライト OFF、書き込みよし
42	"B" } ファイル名
41	"A"
53	"S"
49	"I"
43	"C"
20	
43	"C"
5A	"Z"
38	"8"

コード	意味
46	"F"
42	"B"
30	"0"
31	"1"
53	"S" } ファイル名拡張子（IPL起動型）
79	"y"
73	"s"
20	パスワード無指定
00	ファイル・サイズ
A8	A800Hバイト
00	}

コード	意味
00	} 先頭アドレス 0000H
00	} 実行アドレス
00	0000H
82	82年 } SAVEした日付など。
C1	12月、MON
27	27日
17	17時
58	58分
00	未使用
02	先頭は第2クラスタ
00	未使用

11－6 FAT

ディレクトリ・テーブルとともに、ディスクBASICのファイル管理の要となるのが、**ファイル・アロケーション・テーブル**（File Allocation Table＝ 略して **FAT**） とよばれる領域です。 allocation には「割り当て、配置」の意味があります。

X 1のディスクBASICでは、 FAT はレコード番号14のセクタに記録されていて、ファイルのつながりを管理します。第4節でマスターした方法で、FATをダンプしてみましょう。

《FATを読む》

```
*ME000
:E000=3E01
:E002=110E00
:E005=2100F0
:E008=CDD375
:E00B=C9
:E00C=00
*GE000
*■
```

《FATダンプの例》

```
*DF000
:F000=01 8F 03 04 05 06 07 08 /./......
:F008=09 0A 0B 0C 87 81 0F 82 /........
:F010=89 12 88 14 15 80 83 82 /........
:F018=19 1A 1B 8F 1D 1E 1F 8F /.../....
:F020=21 8E 83 00 00 00 00 00 /!.......
:F028=00 00 00 00 00 00 00 00 /........
:F030=00 00 00 00 00 00 00 00 /........
:F038=00 00 00 00 00 00 00 00 /........
:F040=00 00 00 00 00 00 00 00 /........
:F048=00 00 00 00 00 00 00 00 /........
:F050=8F 8F 8F 9F 8F 8F 8F 8F /////////
:F058=8F 8F 8F 8F 8F 8F 8F 8F /////////
:F060=8F 8F 8F 8F 8F 8F 8F 8F /////////
:F068=8F 8F 8F 8F 8F 8F 8F 8F /////////
:F070=8F 8F 8F 8F 8F 8F 8F 8F /////////
:F078=8F 8F 8F 8F 8F 8F 8F 8F /////////
*

*D
:F080=00 00 00 00 00 00 00 00 /........
:F088=00 00 00 00 00 00 00 00 /........
:F090=00 00 00 00 00 00 00 00 /........
:F098=00 00 00 00 00 00 00 00 /........
:F0A0=00 00 00 00 00 00 00 00 /........
:F0A8=00 00 00 00 00 00 00 00 /........
:F0B0=00 00 00 00 00 00 00 00 /........
:F0B8=00 00 00 00 00 00 00 00 /........
:F0C0=00 00 00 00 00 00 00 00 /........
:F0C8=00 00 00 00 00 00 00 00 /........
:F0D0=00 00 00 00 00 00 00 00 /........
:F0D8=00 00 00 00 00 00 00 00 /........
:F0E0=00 00 00 00 00 00 00 00 /........
:F0E8=00 00 00 00 00 00 00 00 /........
:F0F0=00 00 00 00 00 00 00 00 /........
:F0F8=00 00 00 00 00 00 00 00 /........
*
```

上がレコード番号14のセクタ256バイトの内容(の一例)です。最初の2バイトは(FATやディレクトリをはじめシステムのために) 予約されている値で、3バイト目からの各値がファイルのつながりを示します。

ダンプリストを見ると、最初から80バイトを境にパターンが変わっています。81バイト以降のFATは、実は使用されないことになっていて、そのために「終了コード」8FHが書かれています。129バイト以降は、それさえもなく00Hのままで全く未使用となっています。

さて、FATの有効部分は02H～8FHの値を持ち、次の意味があります。

02H～7FH……そのクラスタに続く、次のクラスタ番号を示す。

80H～8FH……そのクラスタでファイルが終わっていることを示し、7FHを引いた値が最終クラスタ内で使用されるレコード数(＝セクタ数)を意味している。

たとえばFATの3バイト目以降を追うと、

03 → 04 → 05 → 06 → 07 → 08 → 09 → 0A → 0B → 0C → 87

↑ 先頭クラスタは第2クラスタ

↑ 第12クラスタで終了。最終クラスタは、8セクタだけ使用。

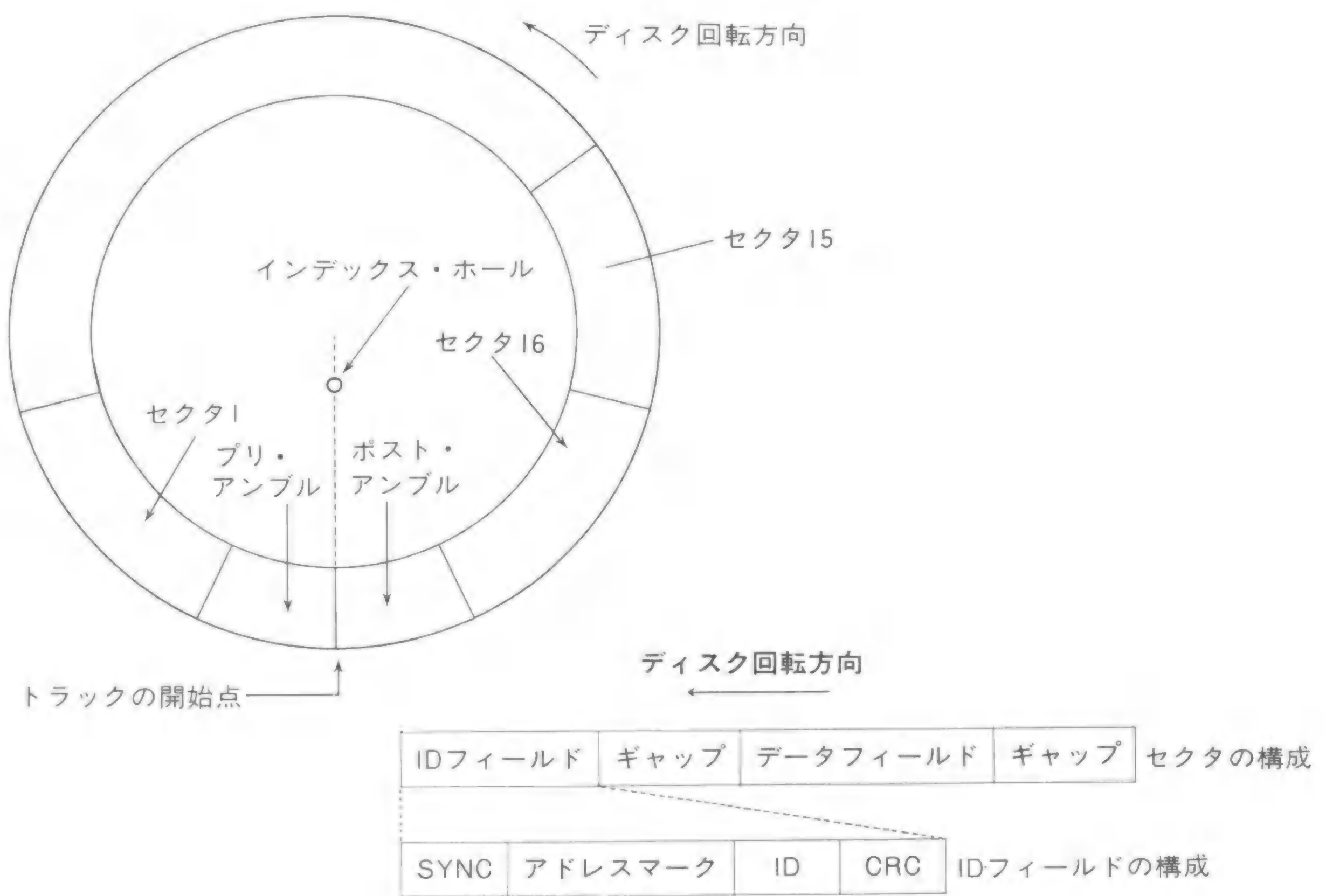
と読むことができて、このファイルは第2クラスタ(先頭セクタはレコード番号32)から始まって、第12クラスタの8レコード(セクタ)を使用して終わっていることがわかります。

このように、ディスクBASICは、ファイルをクラスタ単位で管理しているので、最初の方の少しのセクタしか使用しなくても、1クラスタ分とられてしまいます。

このディスクの FAT をさらに追うと、最後のファイルは、第34クラスタで終了していて、結局使用クラスタはシステム予約分も含め第0クラスタ〜第34クラスタの範囲となります。これは計35クラスタ分で、1枚のディスクには80クラスタ分格納できますから、残りクラスタは80−35=45となります。 FILES コマンドを実行したとき□□clusters free と表示される値はこの値です。

11−7 トラック・フォーマット

1つのトラックは、同心円状の部分ですが、その開始点は中心とインデックスホールを結ぶ延長線上と決められています。トラックにはここから、プリ・アンブル、セクタ1、セクタ2…、セクタ16、ポスト・アンブルの順に書き込まれています。セクタ1とセクタ16の間のアンブル部分は、ディスクの回転ムラなどの不安定要素から生ずる誤差吸収のための領域で一般に**ギャップ（GAP）**とよばれています。（amble というのは「馬の規則的な歩き方」、「ゆっくりした歩調」の意味から来ているようです。 gap は「間隙」の意です。）ギャップは、この他セクタ内部やセクタ間にも設けられています。

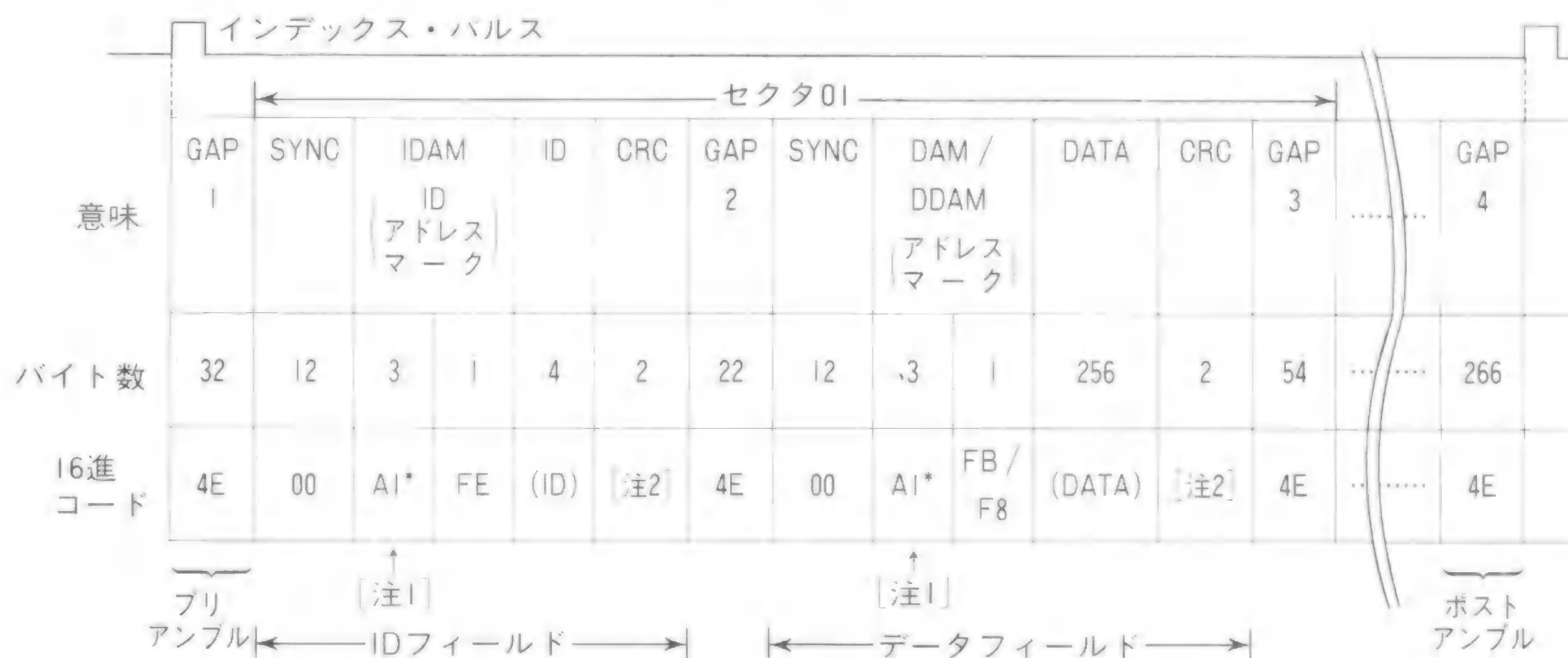
トラックの構成

1つのトラックにデータを書き込む形式（**トラック・フォーマット**）は、IBMの8インチディスクでの標準フォーマットに準じて、5.25インチの倍密ディスクでは次のようになっています（3インチディスクの資料が手許にありませんが、同様のはずです）。

5.25インチ倍密ディスクのトラックフォーマット

インデックス・パルス

セクタ01

	GAP 1	SYNC	IDAM（IDアドレスマーク）		ID	CRC	GAP 2	SYNC	DAM/DDAM（アドレスマーク）		DATA	CRC	GAP 3	……	GAP 4
バイト数	32	12	3	1	4	2	22	12	3	1	256	2	54	……	266
16進コード	4E	00	A1*	FE	(ID)	[注2]	4E	00	A1*	FB/F8	(DATA)	[注2]	4E	……	4E

プリアンブル　[注1]　IDフィールド　[注1]　データフィールド　ポストアンブル

〔注1〕このデータの各バイトは、ミッシング・クロック(第12節参照)を含む。
〔注2〕CRC(Cyclic Redundancy Check)
アドレスマークの先頭ビットから、ID／データの最後のビットまでのビット列を2進数の多項式として、これを$P(X)=X^{16}+X^{12}+X^5+1$で割り算し、その剰余をCRCとしてデータの後へ連続して2バイト書き込む。

各セクタの先頭には、**IDフィールド**とよばれる領域が設けられています。(IDは、identification＝識別 の略ですね。）IDフィールドには、「これからセクタが始まる」ことを示す情報（SYNCやアドレスマーク）と、そのセクタの物理アドレスなど（ID）が記録されています。IDフィールドの末尾には**CRC**とよばれる一種の「チェックサム」が付けられます。

IDは4バイト情報で、

第1バイト＝トラック番号　(00 H～27 H)
第2バイト＝ヘッド番号　(00 H，01 H)
第3バイト＝セクタ番号　(01 H～10 H)
第4バイト＝セクタ長
- 00 H…128 バイト／セクタ
- 01 H…256 バイト／セクタ
- 02 H…512 バイト／セクタ
- 03 H…1024 バイト／セクタ

からなります。第4バイトは、X1では01 H（1セクタ＝256バイト）に設定されます。

IDフィールドの後にはギャップが置かれ、**データフィールド**へと続きます。ここには、SYNC，アドレスマークに続き、実際のデータ(256バイト)が記録され、末尾にデータ部のCRC 2バイトが付きます。

このように見てくると、私たちがセクタとの間で256バイトの情報を正しく読み書きできるように使われる情報（IDやギャップ）が結構場所をとることがわかりますね。ディスクの規格でフォーマット前の記憶容量とフォーマット後の記憶容量に違いが出るのは、このためです。

前節で私たちが実験したようなセクタ単位の通常の読み書きの範囲を越えて、こうしたトラックやセクタの内部構造にまで立ち入るためには、ディスク・ドライブ装置の制御をしているLSI（FDC）の働きを理解しなくてはなりません。次節以降はこの問題を扱います。

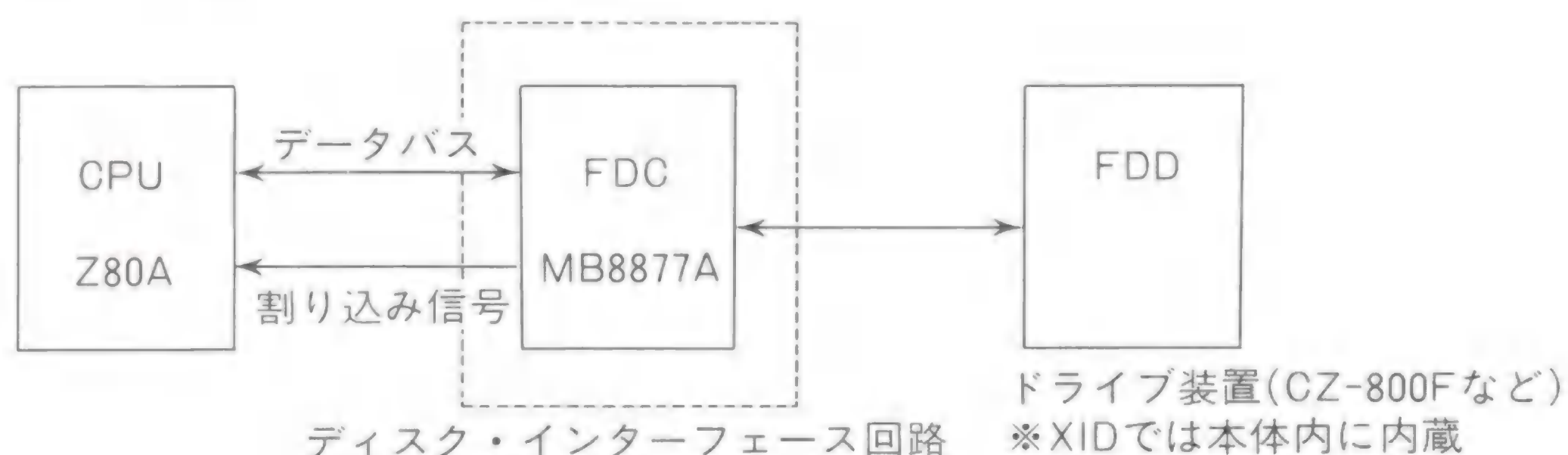
11―8　フロッピー・ディスク・コントローラ（FDC）

X 1本体とフロッピー・ディスク・ドライブ装置（FDD）とのインターフェースにおいて中心的役割を演ずるのが、**フロッピー・ディスク・コントローラ**（**Floppy Disk Controller**＝略して**FDC**とよぶ）とよばれる専用LSIです。X 1においては、

MB8877A

という型番を持つ富士通製のLSIがFDCとして用いられています（X 1 Dでは本体内に標準装備、X 1とX 1 Cでは別売のディスク・インターフェース・カード上に装着されています）。

《**FDCの位置づけ**》

CPU
Z80A
データバス
割り込み信号
FDC
MB8877A
FDD
ディスク・インターフェース回路
ドライブ装置（CZ-800Fなど）
※XIDでは本体内に内蔵

FDCは、CPUから一定のコマンドを与えられると動作を開始し、動作結果を**ステータス**（**status**＝状態）とよばれる1バイト情報の形でCPUに知らせて動作を終了します。CPUはステータスを調べながら、ハンドシェークの形でFDCとのデータ授受を行ないます。（ステータスの詳細は第15節を参照して下さい。）

FDC（MB 8877 A）は、コマンド実行・ステータス表示のために5つのレジスタをユーザーに公開しています。

《MB8877 Aのレジスタ》

レジスタ名	略称	X1でのI／Oアドレス
コマンド・レジスタ	CR	0FF8H番地(OUT命令時)
ステータス・レジスタ	STR	0FF8H番地(IN命令時)
トラック・レジスタ	TR	0FF9H番地
セクタ・レジスタ	SCR	0FFAH番地
データ・レジスタ	DR	0FFBH番地

X 1において、これらのレジスタには、0 FF 8 H～0 FFBH 番地のI／Oアドレスが割りあてられています。コマンドレジスタ（CR）とステータスレジスタ（STR）は同一番地を持ちますが、CPU からの出力命令（OUT）では CR が、入力命令（IN）では STR が選択されます。各レジスタの機能の詳細は、後述するコマンドの説明中で行ないます。

11－9 ドライブ装置のモーター ON ／ OFF

ディスク関係のI／Oアドレスとして、X 1においては、もう1つ次の番地が大切です。

I／Oアドレス	0FFCH番地
機　　能	ドライブ番号(0～3)の選択、ヘッドの選択、各ドライブのモーターON/OFFを行なう。
ビットの意味	ビット7: モーターON/OFF 0＝OFF 1＝ON ビット6: 0 ビット5: 0 ビット4: ヘッド 0＝サイド0 1＝サイド1 ビット3: 0 ビット2: 0 ビット1・ビット0: ドライブ番号 00＝ドライブ0 01＝〃1 10＝〃2 11＝〃3

FDC は、これらの機能を持っていないので、インターフェース・ボード上の別の回路でサポートしています。

さて、ここまで述べたことを利用して、ドライブ装置のモーターを ON ／ OFF し、その時の FDC のステータス変化を見るプログラムが作れます。以下に掲げるものがその一例です。ドライブ装置の電源を入れ、（壊れてもよい）空のディスクを挿入して、プログラムを RUN して下さい。ドライブのモーターを ON した時、ディスクが挿入してあると、ステータスの MSB (Not Ready ビット）が0になるのがわかると思います。CPU はこのビットを見ることで、ドライブの準備状態を知ることができる訳ですね。

リスト　　　　Disk Drive motor ON／OFF

```
100 REM ********************************
110 REM *                              *
120 REM *   Disk Drive motor ON/OFF    *
130 REM *                              *
140 REM *    control port  0FFCH       *
150 REM *                              *
160 REM *                              *
170 REM *        1984.7.3 TUE          *
180 REM *                              *
190 REM ********************************
200 '
210 WIDTH 40
220 CLICK OFF
230 CLS
240 '
250 REM *** instruction ***
260 '
270 CREV 1
280 PRINT "status of FDC"
290 LOCATE 15,3 : PRINT "drive 0 ON ---> 0   KEY"
300 LOCATE 15,4 : PRINT "drive 1 ON ---> 1   KEY"
310 LOCATE 15,5 : PRINT "motor  OFF ---> ESC KEY"
320 CREV 0
330 LOCATE 0,4  : PRINT#0 CHR$(&H1E)
340 LOCATE 0,5  : PRINT "Not Ready"
350 '
360 REM *** main loop ***
370 '
380 A=INP(&HFF8) : REM <--- read status of FDC
390 LOCATE 0,3
400 PRINT RIGHT$("0000000"+BIN$(A),8)
410 LOCATE 0,5
420 IF CHARACTER$(0,3)="0" THEN PRINT "Ready    " ELSE PRINT "Not Ready"
430 '
440 I$=INKEY$(0) : IF I$="" THEN 380
450 IF I$=CHR$(27) THEN 510
460 IF I$="0" OR I$="1" THEN 550
470 GOTO 380
480 '
490 REM === motor OFF ===
500 '
510 OUT&HFFC,N : GOTO 380
520 '
530 REM === drive N motor ON  ===
540 '
550 N=VAL(I$)
560 OUT&HFFC,(&H80 OR N) : GOTO 380
```

11―10　FDCのコマンド

FDC(MB 8877 A) の動作は、CPUから与えられるコマンドにより決定されます。コマンドは、タイプI～IVの4種類に大別されます。

タイプIコマンド……ヘッドの駆動に関するコマンド。
FDDの状態（Ready／Not Ready）に無関係に動作する。

タイプIIコマンド……ディスクのデータ・フィールドに対する書き込みと読み出し動作を指示するコマンド。

タイプIIIコマンド……ディスクのフォーマッティングに関連した特殊コマンド。
現在ヘッドのあるトラックに対して処理を行なう。

タイプIVコマンド……FDCの動作を打ち切ったり、条件により割り込み信号（IRQ）を発生させるためのコマンド。

各々のタイプには1～5個のコマンドがあり、計11個のコマンドが用意されています。各コマンドは、1バイトコードの形で与えられ、その動作の詳細を決めるためのフラグ・オプションを持っています。

FDCコマンド一覧

タイプ	名称	MSB コード LSB	動作概容
I	リストア	0000hV$r_1 r_0$	ヘッドをトラック0へ移動する。
	シーク	0001hV$r_1 r_0$	目的のトラックへ、ヘッドを移動する。
	ステップ	001uhV$r_1 r_0$	ヘッドを1トラック移動する。
	ステップ・イン	010uhV$r_1 r_0$	ヘッドを1トラック内側へ移動する。
	ステップ・アウト	011uhV$r_1 r_0$	ヘッドを1トラック外側へ移動する。
II	リード・データ	100mSEC0	ディスクのデータ(データ・フィールド)を読む。
	ライト・データ	101mSECa_0	ディスク(データ・フィールド)へデータを書く。
III	リード・アドレス	11000E00	ディスクのIDフィールドを読む。
	リード・トラック	11100E00	ディスクの1トラック分の全データを読む。
	ライト・トラック	11110E00	ディスクの1トラック分の全データを書く。
IV	フォース・インタラプト	1101$I_3 I_2 I_1 I_0$	割り込み(IRQ)を発生させる。

オプション・フラグの名称

h　：ヘッド・ロード（Head Load)
V　：ベリファイ（Verify,照合)
r_1, r_0 ：ステップ・レート（Step Rate)
u　：アップ・デート（Update, トラックレジスタの更新)
m　：マルチ・セクタ（multi-Sector)
S　：サイド・ナンバー（Side number)
E　：15 ms ディレイ・イネーブル（15 ms Delay Enable)
C　：コンペア・イネーブル（Compare enable, サイド比較)
a_0　：アドレス・マーク（Address Mark)
I_3～I_0 ：インタラプト・コンディション（Interrupt Condition,IRQ 発生条件)

11—11　タイプ I コマンド

タイプ I コマンドは、ヘッドの駆動に関するコマンドです。フラグ・オプションの意味は次の通りです。

h：ヘッドの上げ下げを指定するフラグ。

h＝1のとき、ヘッド・ロード（ヘッドを下ろし、ディスクに押しつけること）を行なう。

h＝0のとき、ヘッドをディスクから離す。

V：ヘッド移動後、ディスクのトラック番号と、トラック・レジスタの照合（ベリファイ）を行なうかどうか指定するフラグ。

V＝1のとき、IDフィールドのトラック番号と、トラックレジスタの内容を比較する。

V＝0のとき、ベリファイは行なわない。

r_1、r_0、uの各フラグを説明するため「ステップパルス」について述べます。FDCは、ドライブ装置にヘッド移動を指示するために、パルス信号を送ります。これを**ステップパルス**とよび、1トラック分移動させるには1パルスを出力します。さて、ドライブ装置側で機械的に受け入れられるステップパルスの周期（パルスの出力間隔時間）を**ステップレート**とよんでいます。FDCでは、ステップレートを4段階にわたって変えることができます。このために指定されるフラグがr_1，r_0であり、次の意味があります。

《**ステップレート**》

$r_1 r_0$	ステップレート
00	6msec
01	12msec
10	20msec
11	30msec

※ただし、FDCを1MHzのクロックで動作させているときのデータ。

※X1においては、ステップレート20msecが採用されている。
従って、r_1＝1, r_0＝0 を指定する。

最後にuフラグですが、これはステップパルスに伴って、トラック・レジスタの内容の更新を行なうかどうか指定するフラグです。ステップ、ステップ・イン、ステップ・アウトの各コマンドでは、通常 u＝1 と指定し、ステップパルス毎にトラックレジスタの内容を ＋1 または －1 します。 u＝0 と指定すると、FDCからステップパルスを出しても、トラックレジスタは更新されなくなるので特殊用途に使われます（不良トラックがある時の処理など）。

次に、タイプⅠの各コマンドについて概説します。

リストア・コマンド

ディスク・ドライブのヘッドをトラック0に移すコマンドで、コード 00H～0FH で与えられます。X1のディスクBASICを解析すると、リストア・コマンドは、 02H の形で与えられています（ヘッド・ロードなし、ベリファイなし、ステップ・レートは20msec)。データレジスタとトラックレジスタに 00H がセットされてコマンド終了となります。

〔例〕
```
LD   BC,0FF8H ；BC＝コマンドレジスタのアドレス
LD   A,02H    ；リストア・コマンドのコード
OUT  (C),A    ；コマンド出力
```

シーク・コマンド

目的のトラックまで、ヘッドを移動させるコマンドです（seek は「捜し求める」が原義)。通常は、現在ヘッドのあるトラック番号をトラックレジスタに書き込み、目的のトラック番号をデータレジスタに書き込んだ後に、このコマンドを出力すると、シーク動作が正常に行なわれます（ただし上記のレジスタにすでに値がセットしてある時は改めて書き込む必要はありません)。10 H～1 FH のコードを持ち、X 1では 1EH の形で用いることが多いようです（ヘッド・ロードあり、ベリファイあり、ステップ・レート20msec)。このように V＝1 としておくと、シーク完了後、自動的にトラック位置の照合が行なわれ、最初に出会ったIDフィールドのトラック番号とデータレジスタの内容（目的トラック番号）とが比較されます。これらが一致しないときは、後述のステータスレジスタのビット4（Seek Error ビット）がセットされます。

なお、シーク・コマンド終了時、トラックレジスタには、新しいトラック番号がセットされています。

〔例〕 トラック13Hから19Hへのシーク

```
LD   BC,0FFBH  ;BC＝データレジスタのアドレス
LD   A,19H     ;目的トラック番号
OUT  (C),A     ;データレジスタに値をセット
LD   C,0F9H    ;BC＝トラックレジスタのアドレス
LD   A,13H     ;現在ヘッドのあるトラック番号
OUT  (C),A     ;トラックレジスタに値をセット
LD   C,0F8H    ;BC＝コマンドレジスタのアドレス
LD   A,1EH     ;シーク・コマンドのコード
OUT  (C),A     ;コマンド出力
```

ステップ・コマンド

1トラック分ヘッドを移動させるコマンドです。この場合、移動方向（内側、外側）は、コマンド以前に決定された方向によります（後述のステップ・イン、ステップ・アウトの他、リストアやシークを実行しても方向が決定される)。このコマンドが与えられると、ステップパルスが1つ出力されます。

u フラグが1のときは、移動方向が外ならトラックレジスタ（TR）の内容は1減り、方向が内なら TR の内容は ＋1 されます。 u＝0 のときは TR の内容は変化しません。

コード 20 H～3 FH により指定します。

ステップ・イン・コマンド

ヘッドを1トラック分内側（トラック番号増加方向）に移動させるコマンドです。このコマンドが与えられると、ステップパルスが1つ出力されます。

u＝1 のときは、TR の内容は ＋1 され、u＝0 のとき TR は不変です。

コード 40H～5FH により指定します。X1では、ユーティリティプログラム中で、5AH（uフラグ＝1）の形で用いられています。

ステップ・アウト・コマンド

ヘッドを1トラック分外側（トラック番号減少方向）に移動させるコマンドで、これが与えられると、ステップパルスが1つ出力されます。

u＝1 のときは、TR の内容は －1 され、u＝0 のとき TR は変わりません。

コード 60H～7FH により指定します。

11－12 タイプIIコマンド

タイプIIコマンドは、セクタのデータフィールドに対する読み出し、書き込みを行なうコマンドで、最もよく使われます。実行に先立って、対象となるセクタ番号とトラック番号は各々、セクタレジスタ（SCR）、トラックレジスタ（TR）に用意されていなければなりません。

タイプIIコマンドには、m，S，E，C，a_0 のフラグ・オプションがあり、次の意味を持ちます。

m：単一セクタ対象か、連続したセクタ対象かを指定するフラグ。

m＝1のとき、1つのセクタに対してリード／ライト完了後、自動的に SCR の内容を ＋1 し、連続して次のセクタの処理を行なう。

m＝0のとき、1つのセクタに対してのみリード／ライトを行なう。

E：ヘッド・ロード後、その安定状態をチェックするまでに 15 msec の遅れを入れるか否かを指定するフラグ。

E＝1のとき、ヘッド・ロード後、15 msec 経過してから、ヘッドの状態を調べ、安定していれば動作を進める。

E＝0のとき、ヘッド・ロード後、ただちにヘッド状態を調べ、安定してロードされていれば動作を進める。

C：ディスクの ID フィールド内のサイド番号（ヘッド番号）と、Sフラグで指定される値との比較をするか否か指定するフラグ。

C＝1のとき、サイド番号の比較を行なう。

C＝0のとき、サイド番号の比較はしない。

S：サイド番号の比較値を指定する。

S＝1のとき、サイド番号比較値を01Hとする。

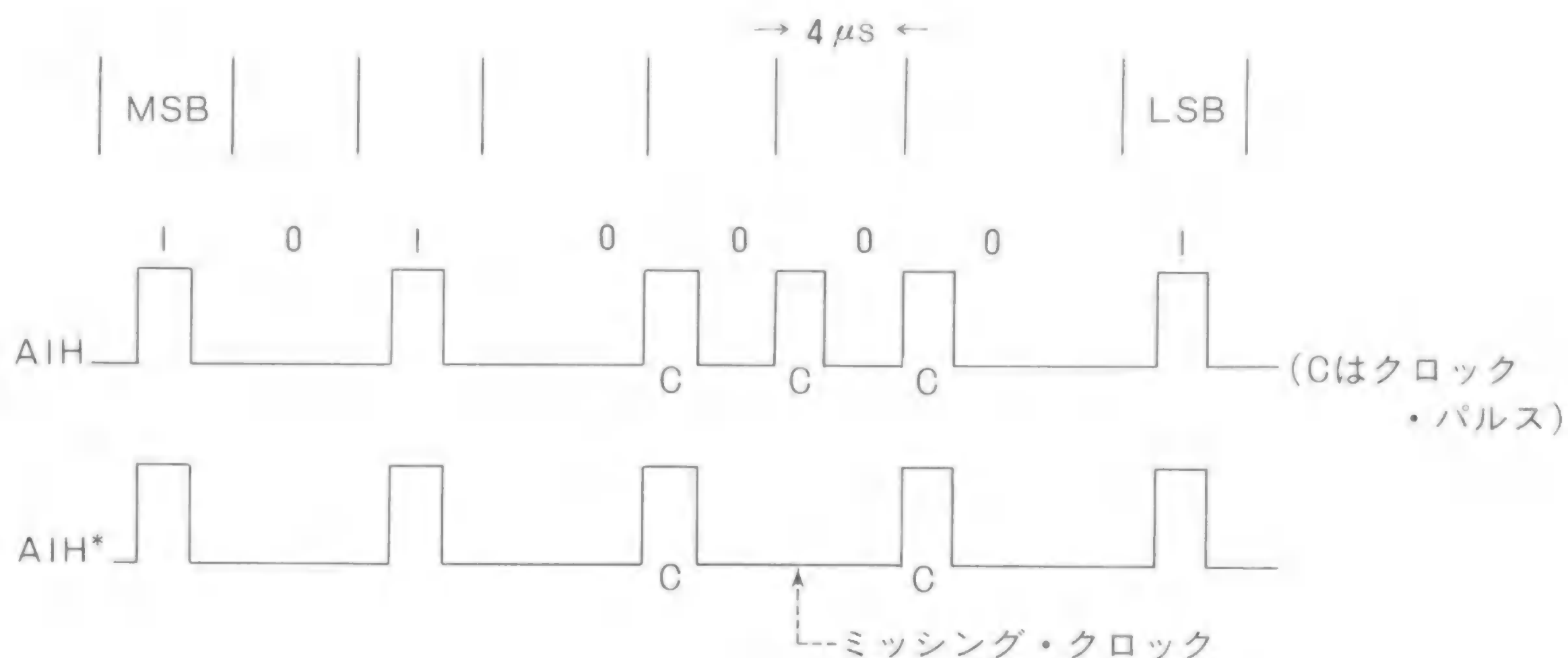
S＝0のとき、サイド番号比較値を00Hとする。

a_0フラグは、ライト・データ・コマンドにおいて指定されます。IBMフォーマットにおいて、各セクタのデータ・フィールドでは、SYNCに続いて、アドレス・マークとよばれる情報が置かれます。通常、アドレス・マークとしては、**データ・アドレス・マーク**（Data Address Mark＝DAM）とよばれる4バイト情報 A1H*,A1H*,A1H*,FBH が書かれ、データ・フィールドの始まりを示します（＊印をつけた情報は、倍密度変調方式でのクロックパルスの入れ方の規則からはずれ、1ビットのクロックパルスが省かれている——ミッシング・クロック——）。しかし、そのセクタのデータを使用しない時には、これとは異なり、**デリーテッド・データ・アドレス・マーク**（Deleted Data Address Mark＝DDAM）を書き込むことにしています。DDAMは、A1H*,A1H*,A1H*,F8H（＊はミッシング・クロックを含む）の計4バイトで構成されます。

さて、a_0フラグですが、ライト・データ・コマンドにおいて、$a_0=1$ と指定すると、データ・フィールドのアドレス・マークとして DDAM が、また $a_0=0$ とすると、DAMが各々書き込まれます。

《ミッシング・クロック》

倍密度変調方式では、1バイト・データA1Hは通常、次図上段のように記録されます。

→ 4 μs ←
MSB
LSB
1 0 1 0 0 0 0 1
A1H
C C C
（Cはクロック・パルス）
A1H*
C C
ミッシング・クロック

これに対して、アドレス・マークにおけるA1H*では、他のデータパターンと区別するために、クロック・パルスが1個省かれています。この省かれたクロック・パルスを**ミッシング・クロック**とよびます。

細かい話が続きましたが、次に、タイプIIの各コマンドについて概説します。

リード・データ・コマンド

セクタ内のデータ・フィールドからデータを読み出すコマンドです。このコマンドが与えられると、FDCは各セクタのIDフィールドをチェックし、目的セクタの検索を始めます。目的セクタが見つかると、データ・フィールドのアドレス・マークをチェックし、もしDDAMなら、ステータスのビット5(Record Type ビット)を1として、そのセクタが使用されないことをCPUに知らせます。

また、アドレス・マークがDAMなら、FDCはデータの読み取りにかかります。その後、FDCは1バイトをディスクから読み取るごとに、ステータスのビット1(Data Request ビット)を1とし、CPUにデータの読み取りを要求します。このビットを見ながら、CPUはデータレジスタを読み出すことで、1バイトずつデータをディスクから受け取っていく訳です。もし、一定時間内にCPUがデータレジスタを読み出さないと、ステータスのビット2(Lost Data ビット)がセットされ、データが失われたことをCPUに知らせた後、データレジスタには次の新しいデータが用意されます。

IDフィールドに記録されているセクタ長(X1では256バイト)の読み込みが終了すると、CRCデータのチェックをし、正しく読み込まれていることが確認されたら、mフラグに従って、終了するか、次のセクタの処理に進みます。CRCが一致しない時は、ステータスのビット3(CRC Error ビット)をセットし、mフラグの状態にかかわらずコマンドを終了します。

リード・データ・コマンドのコードは、80H～9EHの範囲の偶数値(LSB=0)です。HuBASICのディスクからのリード・データ・ルーチン内では、80Hの形で用いられます(単一セクタ,サイド番号比較なし、ディレイなし)。

ライト・データ・コマンド

FDCのレジスタで指定された、目的のセクタのデータ・フィールドにデータを書き込むコマンドです。このコマンドの動作時、FDCは、データ・フィールドのSYNCから書き始め、CRCを書き込んだ後に、GAPとして1バイトFFHを書き込んで終了します。(読み出し時の最後のビットを正しくするためのものです。)

CPUからFDCに送信すべきデータは、データ・フィールドの本体をなす256バイト(IDフィールド内のセクタ長指定バイトで決定)です。FDCは、これらをディスクに書き込むと、その後にデータから計算される2バイトのCRCを自動的に書き込みます。

ライト・データ・コマンドのコードは、A0H～BFHで与えられます。コードのLSBはa_0フラグであって、通常は$a_0=0$として使用し、アドレス・マークにはDAM(Data Address Mark)が書き込まれます。ある特定のセクタに印をつけたい時は、そのセクタに書き込む時に$a_0=1$とすれば、アドレス・マークとしてDDAM(Deleted Deta Address Mark)が書き込まれます。このマークがつくと、後にセクタを読み出した時に、ステータスのRecord type ビットが1になるので、マーク付を検出できます。プログラム上でこのようなセクタを読めなくしておけば、一種のプログラム保護(プロテクト)や不良セクタ処理に利用できる訳ですね。第17節でこの問題を扱います。

HuBASICのライト・データ・ルーチン内では、 A0H の形で用いられています（単一セクタ、サイド比較なし、$a_0 = 0$）。

11－13 タイプIIIコマンド

タイプIIIコマンドは、現在ヘッドのいるトラックに対して処理を行なうコマンドであって、ディスクのトラック・フォーマットと深く関係しています。Eフラグを持ちますが、これはタイプIIコマンドでのEフラグと同様です。

リード・アドレス・コマンド

IDフィールドを読み出すコマンドですが、読み出されるIDフィールドは、FDCが読み込みを開始してから最初に見つけたIDフィールドだけです。読み出されるデータは、IDとCRCの計6バイトです。このコマンドでは、トラック番号がセクタレジスタに保存されます。IDフィールドを読み出した結果、CRCエラーが生じても自動的に再動作する事はないので、必要ならコマンドを再び与えなくてはなりません。

コマンドのコードは C0H(Eフラグ＝0)、C4H(Eフラグ＝1) ですが、HuBASICの内部ルーチンで使われている形跡はありません。

リード・トラック・コマンド

トラックの記録内容をGAP,SYNC等を含めすべて読み出すコマンドです。

コマンドのコードは E0H(Eフラグ＝0)、E4H(Eフラグ＝1) のいずれかで指定します。HuBASICでは使われている例がありません。

ライト・トラック・コマンド

トラックのすべてのデータを書くコマンドです。この操作は、**トラック・イニシャライズ**あるいは**フォーマッティング**とよばれます。

ライト・トラック・コマンドでは、ディスクに書き込む1トラック分のデータをすべてCPUからFDCに送信する必要があります。このコマンドを用いれば、書き込みに関しては自由なフォーマットを使用できます。

FDCがこのコマンドを受けると、すぐに1バイト目のデータ要求が出されるので、3バイト以内にCPUからデータを与えないと、 Lost Data のステータス・ビットを立ててコマンドは終了します。この3バイト以内に最初のデータがCPUから送られてくると、FDCはインデックス・パルスを待って、次のデータ要求を出すとともに、先の1バイト目がディスクに書き込まれます。データの転送は、次のインデックス・パルスが来るまで続けられます。

この際、トラック内のデータとして、ミッシング・クロックを含むものや、CRCなどもあるので、それらの処理のため、CPUからFDCへ送信されるデータには「特別の意味」を持たせてあります。次図がそれらをまとめたものです。

ライト・トラック時のデータとその意味

DRに書き込まれる内容	実際に書き込まれる内容	意味	
		データの意味	FDCの内部動作
00H ≀ F4H	00H ≀ F4H	通常のデータ	――――
F5H	A1H*	IDアドレス・マーク、データ・アドレスマークの前提データで、ミッシング・クロックを含む。	CRCジェネレータをプリセットする。
F6H	C2H*	(8インチディスク以外は未使用)	――――
F7H	2バイトのCRC	内部で作成された2バイトのCRC	ディスクへのCRC書き込みをスタートする。
F8H ≀ FFH	F8H ≀ FFH	通常のデータ	――――

ライト・トラック・コマンドのコードは、F0H(Eフラグ=0)、F4H(Eフラグ=1)のいずれかで指定されますが、X1においては、ディスク初期化のユーティリティ・プログラム「Utility.Obj」内で、F4Hの形で用いられています。

11—14 タイプⅣコマンド

タイプⅣコマンドは、**フォース・インタラプト・コマンド**とよばれ、FDCの動作を強制的に打ち切るため、および条件により割り込み要求信号(IRQ)を発生させるためのコマンドです。IRQ発生条件は、次のフラグにより設定されます。

《**Iフラグ**》

フラグ	IRQ発生条件
I_0	READY入力の立ち上がりでIRQ="H"
I_1	READY入力の立ち下がりでIRQ="H"
I_2	各インデックス・パルスでIRQ="H"
I_3	無条件で即IRQ="H"

タイプⅠ～Ⅲのコマンドが、FDCのアイドル状態(Busy=0のとき)にのみ与える事ができるのに対し、タイプⅣコマンドは、FDCの状態に関係なく与えることのできる唯一のコマンドです。タイプⅣコマンドが与えられると、FDCは現在実行中のコマンドを打ち切って、I_0～I_3の条件に従って、割り込み要求信号(IRQ)を発生します。

通常のコマンドにより発生したIRQは、ステータスレジスタのリードもしくは、コマンドの書き込みによりリセットされますが、タイプⅣコマンドにより発生したIRQは、I_0～I_3＝0として新たにタイプⅣコマンドを与えた後、ステータスレジスタのリードもしくは、コマンドの書き込みによりリセットされます。

コマンドはコード D0H～DFH のいずれかで指定され、X1においては、「Utility.Obj」の中で、FDCの動作打ち切り用に D0H（I_0～I_3＝0）の形で用いられています。

11－15 ステータス

ステータス（status）とは、コマンド動作の進行状況、終了状態およびドライブの状態を示すために、FDCが出力する情報（1バイト）で、FDCのステータスレジスタ STR には、実行中あるいは終了したコマンドのステータスが出力され、各々のビット STRn はコマンドにより意味が異なります。

ステータス一覧表

コマンド ＼ ステータス・レジスタ		STR_7	STR_6	STR_5	STR_4	STR_3	STR_2	STR_1	STR_0
タイプⅠ	全コマンド	Not Ready	Write Protect	Head Engaged	Seek Error	CRC Error	Track00	Index	Busy
タイプⅡ	リード・データ	Not Ready	0	Record Type	Record Not Found	CRC Error	Lost Data	Data Request	Busy
	ライト・データ	Not Ready	Write Protect	Write Fault	Record Not Found	CRC Error	Lost Data	Data Request	Busy
タイプⅢ	リード・アドレス	Not Ready	0	0	Record Not Found	CRC Error	Lost Data	Data Request	Busy
	リード・トラック	Not Ready	0	0	0	0	Lost Data	Data Request	Busy
	ライト・トラック	Not Ready	Write Protect	Write Fault	0	0	Lost Data	Data Request	Busy
タイプⅣ	（他のコマンド実行中の時）	（今まで実行していたコマンドのステータス・ビットと同様の意味）							0
	（実行中のコマンドなしの時）	Not Ready	Write Protect	Head Engaged	0	0	Track00	Index	0
マスタ・リセット		（タイプⅠコマンドに準ずる）							

ステータス・ビットの意味は次の通りです。

ステータス名称	意　　　　　味
Not Ready	1で、ドライブが動作可能状態(Ready)でないことを示す。
Write Protect	1で、ディスクへの書き込みが禁止されていることを示す。
Head Engaged	1で、ヘッドがディスクに押しつけられていることを示す。
Seek Error	1で、ベリファイ動作が不成功だったことを示す。
CRC Error	1で、IDフィールド、データフィールドの読み出し時に、エラーがあったことを示す(CRCデータの不一致)。
Track 00	1で、ヘッドがトラック0にあることを示す。
Index	1で、インデックス・ホールを検出したことを示す。
Busy	1で、FDCがコマンド動作中であることを示す。
Record Type	リード動作時に、データフィールドのアドレス・マークがDDAM(Deleted Data Address Mark)のとき1となり、DAM(Data Address Mark)のとき0となる。
Write Fault	ライト動作時に、1で、動作が打ち切られたことを示す。
Record Not Found	1で、「指定されたトラック番号、サイド(ヘッド)番号、セクタ番号を持ち、正しく読み出されたIDフィールド」がなかったことを示す。
Lost Data	1で、所要時間内に、データレジスタの読み出し、あるいは書き込みが行なわれなかったことを示す。
Data Request	1で、FDCがCPUに対して、データレジスタの読み出し、あるいは書き込みを要求していることを示す。

11−16 ディスク関係システム・サブルーチン

第4節で「リード」と「ライト」の2つのシステム・サブルーチンを紹介しましたが、本節では、これらルーチンの構成要素でもあるさらに基本的なサブルーチンを紹介します。

名　　称	物理アドレス算出ルーチン
アドレス	75A3H
入力条件	DEレジスタ＝レコード番号 Aレジスタ＝使用セクタ数 (74BEH) ＝ドライブ番号(0～4)
出力条件	Dレジスタ＝トラック番号＊2＋ヘッド番号 Eレジスタ＝セクタ番号 Aレジスタ＝　　〃 A'レジスタ＝使用セクタ数 HLレジスタ、BCレジスタは保存される。
機　　能	DEレジスタのレコード番号から、トラック番号とセクタ番号を算出する。ドライブ番号が4以上のときは、Bad file descriptorエラー、レコード番号が0500H以上のときは、Bad recordエラーとなる。

名　　称	ドライブ・モーターON
アドレス	751EH
入力条件	Dレジスタ　＝トラック番号＊2＋ヘッド番号 (74BEH)　＝ドライブ番号
出力条件	(74BEH)　＝80H＋ヘッド番号＊10H＋ドライブ番号 Dレジスタ　＝トラック番号 HL、Eは保存。
機　　能	指定ドライブのモーターをONし、ヘッドを選択する。Dレジスタにトラック番号をセットする。 ドライブ装置の電源がOFFか、ディスクが挿入されてなければ、Device offline エラーとなる。

名　　称	シーク
アドレス	757DH
入力条件	Dレジスタ　＝目的トラック番号 (74BEH)　＝下位4ビットにドライブ番号
出力条件	指定ドライブのトラック番号ワークエリア（74C3H～74C6H）を更新する。
機　　能	Dレジスタの指定するトラックまで、指定ドライブのヘッドをシークする。

名　　　　称	リストア・シーク
アドレス	7551H
入力条件	Dレジスタ　＝目的トラック番号 (74BEH)　＝下位4ビットにドライブ番号
出力条件	指定ドライブのトラック番号ワークエリア（74C3H〜74C6H）を更新する。
機　　　　能	リストア後、Dレジスタの指定するトラックまで、ヘッドをシークする。

名　　　　称	FDCコマンド出力
アドレス	763CH
入力条件	Aレジスタ　＝FDCへ出力するコマンド Eレジスタ　＝セクタ・レジスタに設定する値
機　　　　能	Eレジスタの値をFDCのセクタ・レジスタにセットしたのち、AレジスタのコマンドをFDCに出力する。実行後は割り込み禁止となる。

名　　　　称	ドライブ・モーターOFF
アドレス	764CH
入力条件	(74BEH)　＝下位2ビットにドライブ番号
機　　　　能	指定ドライブのモーターをOFFする。

11−17　FDCを操る（デリーテッド・データ・マーク）

第4節で紹介した「ディスクへのライト・データ」のシステムサブルーチンの最初の方を逆アセンブルしたものが次のリストです。

《システムサブルーチン「ライト・データ」》

```
765A 32C376      LD    (76C3H),A
765D ED53C576    LD    (76C5H),DE
7661 22C876      LD    (76C8H),HL
7664 CDA375      CALL  75A3H
7667 CD1B77      CALL  771BH
766A CD1E75      CALL  751EH
766D CD7D75      CALL  757DH
7670 3E05        LD    A,05H
7672 F5          PUSH  AF
7673 E5          PUSH  HL
7674 3EA0        LD    A,0A0H
7676 CD3C76      CALL  763CH
```

前節で紹介した基本ルーチンが利用されていますね。これ以外に 765 AH～7661 H では後にベリファイ動作をするためワークエリアにデータをセットしています。また、 771 BH のサブルーチンでは、ライト・プロテクト・ファイルでないことのチェックをします。さて、 7674 H～7678 H が、FDC コマンド A 0 H(ライト・データ) を FDC に出力する部分です。ここで、a_0フラグを 1 にしてやる(コマンドを A 1 H とする)と、データ・フィールドに、デリーテッド・データ・マーク(DDAM)が書かれ、読み出し時に、 Record type ステータス・ビットで検出できるはずです。

第 4 節で用いたレコード番号 04 FFH のセクタに DDAM をつけてみましょう。例の「壊れてもよいディスク」をドライブ 0 に挿入し、モニターで 7675 H 番地を A 1 H に変更してから、「ライト・データ」を実行します。(F 000 H～F 0 FFH には第 4 節と同じデータを用意して下さい！)

《ライト・データでa_0＝1にする》

```
MON
*M7675
:7675=A1
:7676=CD
*ME000
:E000=3E01
:E002=11FF04
:E005=2100F0
:E008=CD5A76
:E00B=C9
:E00C=00
*GE000
*■
```

書き込みが終わったら、忘れないうちに 7675 H 番地を A 0 H に戻しておきましょう。

さて、このセクタをシステムサブルーチン 75 D 3 H(リード・データ) で読み出すと何の異常もなく読み出せます。これは、「リード・データ」ルーチン内で、 Record type ステータスの検出をしていないからです。最初の方を逆アセンブルすると次のようになっています。前節で紹介したサブルーチンが見えますね。

《システムサブルーチン「リード・データ」》

```
75D3 CDA375     CALL 75A3H
75D6 CD1E75     CALL 751EH
75D9 CD7D75     CALL 757DH
75DC 3E05       LD   A,05H
75DE F5         PUSH AF
75DF E5         PUSH HL
75E0 3E80       LD   A,80H
75E2 CD3C76     CALL 763CH
75E5 D5         PUSH DE
75E6 11FBF8     LD   DE,0F8FBH
75E9 4B         LD   C,E
75EA ED78       IN   A,(C)
75EC 4A         LD   C,D
75ED ED78       IN   A,(C)
75EF 0F         RRCA
75F0 300B       JR   NC,75FDH
75F2 0F         RRCA
75F3 30F8       JR   NC,75EDH
75F5 4B         LD   C,E
75F6 ED78       IN   A,(C)
75F8 77         LD   (HL),A
75F9 23         INC  HL
75FA 4A         LD   C,D
75FB 18F0       JR   75EDH
```

75 F 5 H～75 F 9 H 番地が読み出したデータをメモリーに格納する部分です。ここを次のように変更して、ステータスをメモリーに格納した後、BASIC のホットスタート 14 CCH 番地にジャンプするよう変更してみましょう。

アドレス	元のコード	変更後	変更後の意味
75F5H	4B	ED	IN A,(C) ステータスを読む
75F6H	ED	78	
75F7H	78	77	LD (HL),A メモリーに格納
75F8H	77	C3	JP 14CCH BASIC ホットスタートへジャンプ
75F9H	23	CC	
75FAH	4A	14	

さていよいよ、レコード番号 04 FFH のセクタを読んで、ステータスの Record type ビット（ビット 5）が立っているかどうか検証しましょう。

次のプログラムを打ち込んで下さい。HL レジスタを E 200 H にセットしましたから、ステータスは E 200 H 番地に格納されるはずですね。

```
MON
*ME100
:E100=3E01
:E102=11FF04
:E105=2100E2
:E108=CDD375
:E10B=C9
:E10C=00
*GE100
```

* GE 100 で実行すると、ディスクが動き出し、画面クリア後 Ok が表示され、BASIC のコマンド待ちに戻ったことがわかります。では、「高鳴る胸をおさえて……」E 200 H 番地をダンプしてみましょう。

《ビット 5 は立っていた！》

```
Ok
MON
*DE200
:E200=23 00 00 00 00 00 00 00 /#.......
:E208=00 00 00 00 00 00 00 00 /........
:E210=00 00 00 00 00 00 00 00 /........
:E218=00 00 00 00 00 00 00 00 /........
:E220=00 00 00 00 00 00 00 00 /........
:E228=00 00 00 00 00 00 00 00 /........
:E230=00 00 00 00 00 00 00 00 /........
:E238=00 00 00 00 00 00 00 00 /........
:E240=00 00 00 00 00 00 00 00 /........
:E248=00 00 00 00 00 00 00 00 /........
:E250=00 00 00 00 00 00 00 00 /........
:E258=00 00 00 00 00 00 00 00 /........
:E260=00 00 00 00 00 00 00 00 /........
:E268=00 00 00 00 00 00 00 00 /........
:E270=00 00 00 00 00 00 00 00 /........
:E278=00 00 00 00 00 00 00 00 /........
*■
```

実験成功です。ステータスは

```
23H  =  |0|0|1|0|0|0|1|1|
             ↑       ↑ └ Busy
      Record type    └ Data Request
```

となって見事 DDAM(デリーテッド・データ・マーク) が検出されました。そうそう、忘れぬうちに 75 F 5 H～75 FAH 番地の変更箇所を元に戻しておいて下さい。

HuBASIC では、 Record type ビットの検出をしていませんが、システム・プログラムがこのビットを検出するようになっていれば、書き込み時に a_0フラグを 1 にすることが意味を持ってきます。すなわち、オリジナル・ディスクでは特定のセクタを $a_0=1$ で書き込んでおきます。このディスクがコピーされたとすると、コピー・ルーチンでは通常 $a_0=0$ でコピーしますから、特定のセクタのアドレス・マークは DAM になっています。そこで、システム内で特定のセクタを読んだ時に Record type ビットを見て 0 なら、コピー・ディスクだと判断できる訳です。これを利用すると、ディスクのコピー禁止（プロテクト）を行なうことができそうですね。

このように、FDC の働きを理解すると BASIC では思いもよらないことが実現できます。本章は FDC についてかなり詳しく説明したつもりですから、皆さんも FDC の制御をいろいろ研究してみて下さい。きっと興味深い発見があることでしょう。

第12章 SHARP HuBASIC

本章では、ＸⅠシリーズの性能をひき出す素晴らしい基本ソフトウェアである「SHARP HuBASIC」に関して、マニュアルに記載されていない部分を中心にして述べます。

12－1 メモリー・マップ

まず、SHARP HuBASIC(以下 HuBASIC と略称します)を起動した直後のメモリー・マップを見てみます。BASIC マニュアルに書かれているマップに、必要なアドレス値を書き加えてみました。これらはコマンドやプログラム動作などに伴い、以後、動的に変化していきます。

HuBASIC起動時メモリー・マップ

テープ	ディスク	メモリー	備考
0000	(0000)	IOCS〔注〕	〔注〕Input Output Control System の略で、入出力制御の基本サブルーチン群からなる。
0FE2	(0FE2)	モニター	
14A0	(14A0)	BASIC インタプリタ	
9FC4	(A8A7)		
9FC5	(A8A8)	テキスト	←NEW ON で指定したテキスト・アドレス
		テキスト・エンド・コード "00 00"	
9FC7	(A8AA)	変数エリア	←VARPTR
		変数エンド・コード "00"	
9FC8	(A8AB)	ファイル用ストリングバッファ #0、#1、…	←STRPTR MAXFILESnでとられるエリア (n＋1)×(16＋256)バイト
A1E8	(ABDB)	ストリング・データバッファ	
A1EA	(ABDD)	テンポラリー・ストリング エリア	
A1EB	(ABDE)	↓ フリー・エリア ↑	
FDFF	(FDFF)		
FE00	(FE00)	BASIC用スタック	
FE00	(FE00)	FAC	← (CLEAR、LIMIT) −& H100
		ユーザー用マシン語フリー・エリア	←CLEAR、LIMIT で指定したアドレス
FF00	(FF00)	IPL、モニター用ワーク・エリア	
FFFF	(FFFF)		

〔注〕テープBASIC(CZ-8 CB 01)と、ディスクBASIC(CZ-8 FB 01)の場合を掲げました。アドレス値は両者で少し異なります。図中括弧でくくった方が、ディスクBASICのアドレスです。

12―2 テキストの格納形式

ユーザーが作成するBASICプログラムのテキストは、テキストエリア9FC5H番地以降(ディスクBASICでは、A8A8H番地以降)に格納されます。

この様子を見るために、次の簡単なプログラムについて、そのテキスト・エリアへの格納形式をダンプしてみます(ディスクBASICの方は、一度モニターで ＊G 14A0 によりBASICをコールドスタートさせ、メモリー内をクリアしてから、プログラムを入力するとよいでしょう)。モニターにより、テープBASICの方は ＊D 9FC5 ディスクBASICの方は ＊D A8A8 として、テキストエリアをダンプすると以下のリストのようになるはずです。

テキスト格納形式

```
10 INPUT A
20 INPUT B
30 PRINT A*B
40 END
```

《テープBASICの場合》

```
:9FC5=08 00 0A 00 91 20 41 00 /....| A.
:9FCD=08 00 14 00 91 20 42 00 /....| B.
:9FD5=0A 00 1E 00 8F 20 41 FC /..../ A円
:9FDD=42 00 06 00 28 00 98 00 /B...(.┘.
:9FE5=00 00 00 00 4F E5 5D 00 /....O♣].
:9FED=00 00 00 00 00 00 00 00 /........
:9FF5=00 00 00 00 00 00 00 00 /........
:9FFD=00 00 00 00 00 00 00 00 /........
:A005=00 00 00 00 00 00 00 00 /........
:A00D=00 00 00 00 00 00 00 00 /........
:A015=00 00 00 00 00 00 00 00 /........
:A01D=00 00 00 00 00 00 00 00 /........
:A025=00 00 00 00 00 00 00 00 /........
:A02D=00 00 00 00 00 00 00 00 /........
:A035=00 00 00 00 00 00 00 00 /........
:A03D=00 00 00 00 00 00 00 00 /........
```

《ディスクBASICの場合》

```
:A8A8=08 00 0A 00 91 20 41 00 /....| A.
:A8B0=08 00 14 00 91 20 42 00 /....| B.
:A8B8=0A 00 1E 00 8F 20 41 FC /..../ A円
:A8C0=42 00 06 00 28 00 98 00 /B...(.┘.
:A8C8=00 00 00 00 4F 3A 69 00 /....O:i.
:A8D0=00 00 00 00 00 00 00 00 /........
:A8D8=00 00 00 00 00 00 00 00 /........
:A8E0=00 00 00 00 00 00 00 00 /........
:A8E8=00 00 00 00 00 00 00 00 /........
:A8F0=00 00 00 00 00 00 00 00 /........
:A8F8=00 00 00 00 00 00 00 00 /........
:A900=00 00 00 00 00 00 00 00 /........
:A908=00 00 00 00 00 00 00 00 /........
:A910=00 00 00 00 00 00 00 00 /........
:A918=00 00 00 00 00 00 00 00 /........
:A920=00 00 00 00 00 00 00 00 /........
```

メモリーに格納されている各コードの意味は次の通りです（テープBASICのアドレスで説明します)。

テキストエリア内コードの意味

行番号	アドレス	コード	意味
10行	9FC5	08	0008…1行バイト数（8バイト）
	9FC6	00	
	9FC7	0A	000A…行番号10
	9FC8	00	
	9FC9	91	INPUT の中間コード
	9FCA	20	スペース・コード
	9FCB	41	変数名Aのコード
	9FCC	00	1行のエンドマーク
20行	9FCD	08	0008…1行バイト数（8バイト）
	9FCE	00	
	9FCF	14	0014…行番号20
	9FD0	00	
	9FD1	91	INPUT の中間コード
	9FD2	20	スペース・コード
	9FD3	42	変数名Bのコード
	9FD4	00	1行のエンドマーク
30行	9FD5	0A	000A…1行バイト数（10バイト）
	9FD6	00	
	9FD7	1E	001E…行番号30
	9FD8	00	
	9FD9	8F	PRINT の中間コード
	9FDA	20	スペース・コード
	9FDB	41	変数名のAコード ┐
	9FDC	FC	*の中間コード │ A*Bの意味
	9FDD	42	変数名Bのコード ┘
	9FDE	00	1行のエンドマーク
40行	9FDF	06	0006…1行バイト数（6バイト）
	9FE0	00	
	9FE1	28	0028…行番号40
	9FE2	00	
	9FE3	98	END の中間コード
	9FE4	00	1行のエンドマーク

9 FE 4 H 番地までが、BASICプログラムの10〜40行の格納部分です。9 FE 5 H〜9 FE 6 H 番地に 00 H が2つ続きますが、これは、テキストの終了を意味するコードです(メモリー・マップ参照)。9 FE 7 H 番地の 00 H は変数エリアの終了コードです。

さて、9 FE 8 H 番地から妙なコードが出現しますが、これは「ファイル用ストリング・バッファ」といって、第11節で再び触れますが、次の意味があります。

《ファイル用ストリング・バッファ＃0の内容》

アドレス	コード	意　　味
9 FE 8	00	ファイル属性（00HはKILLされたファイルを意味する。）
9 FE 9	4F	ファイル・モード（4 FHは"O"＝OUTPUTモード）
9 FEA 9 FEB	E5 5D	5 DE 5…デバイス・テーブル・アドレス（この場合はプリンタ指定）
⋮	⋮	⋮

（以　下　略）

〔注〕ファイル用ストリング・バッファは起動後すぐにプログラムを入力した時点では現われませんが、一度でも LIST をとったりすると出現します。上では LLIST をとったために、デバイス指定がプリンタ（LPT）になり、テーブル・アドレスが 5 DE 5 H になっています。皆さんが画面出力している時には、テーブル・アドレスは 5 D 39 H(デバイス CRT 指定) になっていると思います。

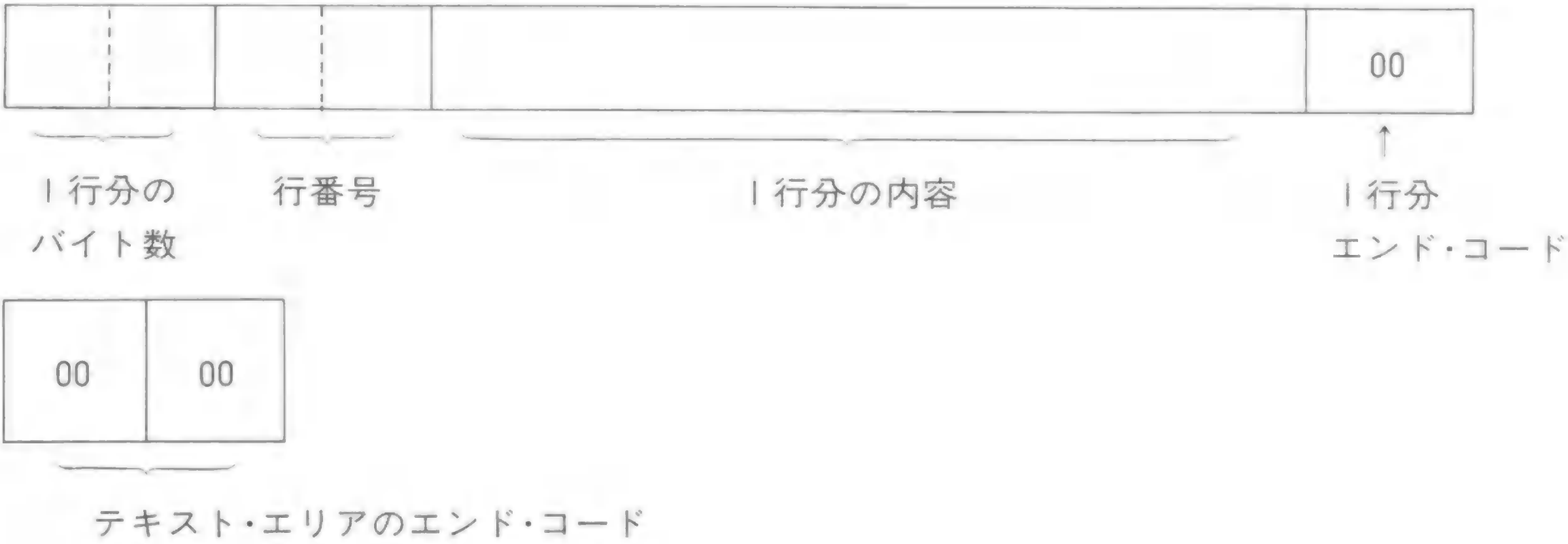
メモリーの内容を見ていただくとわかるように、BASICプログラムの1つの行(行番号で始まる)の格納形式は以下にようにまとめられます。

プログラム1行分の格納形式

1行分のバイト数	行番号	1行分の内容	00

00：1行分エンド・コード

00	00

テキスト・エリアのエンド・コード

12−3 中間コード

前節でもわかるように、BASICコマンド INPUT などはそのままの形で格納されている訳ではありません。HuBASICで使用するこのような**予約語**（key word, reserved word）は、メモリーの節約と処理速度を高めるために、1〜2バイトに短縮されて「中間コード」(token) とよばれる形で、テキストエリアに格納されています。HuBASICにおける予約語と中間コードの対応は次のように決められています。

===== TOKEN TABLE (1) =====

KEY WORD	TOKEN	KEY WORD	TOKEN	KEY WORD	TOKEN
GOTO	80H		ABH	LFILES	D6H
GOSUB	81H		ACH	DEVICE	D7H
GO	82H		ADH	NAME	D8H
RUN	83H	DEFINT	AEH	KILL	D9H
RETURN	84H	DEFSNG	AFH	LSET	DAH
RESTORE	85H	DEFDBL	B0H	RSET	DBH
RESUME	86H	DEFSTR	B1H	INIT	DCH
LIST	87H	DEF	B2H	VDIM	DDH
LLIST	88H		B3H	MAXFILES	DEH
DELETE	89H	LOAD	B4H		DFH
RENUM	8AH	SAVE	B5H	TO	E0H
AUTO	8BH	MERGE	B6H	STEP	E1H
EDIT	8CH	CHAIN	B7H	THEN	E2H
FOR	8DH	CONSOLE	B8H	USING	E3H
NEXT	8EH	WIDTH	B9H	SUB	E4H
PRINT	8FH	OUT	BAH	BASE	E5H
LPRINT	90H	SEARCH	BBH	TAB	E6H
INPUT	91H	WAIT	BCH	SPC	E7H
LINPUT	92H	PAUSE	BDH	EQV	E8H
IF	93H	WRITE	BEH	IMP	E9H
DATA	94H	SWAP	BFH	XOR	EAH
READ	95H	ERASE	C0H	OR	EBH
DIM	96H	ERROR	C1H	AND	ECH
REM	97H	ELSE	C2H	NOT	EDH
END	98H	CALL	C3H	><	EEH
STOP	99H	MON	C4H	<>	EFH
CONT	9AH	LOCATE	C5H	=<	F0H
CLS	9BH	SCREEN	C6H	<=	F1H
CLEAR	9CH	KEY	C7H	=>	F2H
ON	9DH		C8H	>=	F3H
LET	9EH		C9H	=	F4H
NEW	9FH	LABEL	CAH	>	F5H
POKE	A0H	RANDOMIZE	CBH	<	F6H
OFF	A1H	OPTION	CCH	+	F7H
WHILE	A2H	LINE	CDH	-	F8H
WEND	A3H	OPEN	CEH	MOD	F9H
REPEAT	A4H	CLOSE	CFH	¥	FAH
UNTIL	A5H	SIZE	D0H	/	FBH
	A6H	FIELD	D1H	*	FCH
	A7H	GET	D2H	^	FDH
	A8H	PUT	D3H		
TRON	A9H	SET	D4H		
TROFF	AAH	FILES	D5H		

===== TOKEN TABLE (2) =====

KEY WORD	TOKEN	KEY WORD	TOKEN	KEY WORD	TOKEN
WINDOW	FE80H	LAYER	FE90H	EFFECT	FEA0H
PSET	FE81H	CANVAS	FE91H	GRAPH	FEA1H
PRESET	FE82H	CREV	FE92H	MUSIC	FEA2H
COLOR	FE83H	CFLASH	FE93H	TEMPO	FEA3H
CIRCLE	FE84H	CGEN	FE94H	CURSOR	FEA4H
POLY	FE85H	CSIZE	FE95H	VERIFY	FEA5H
PAINT	FE86H	EJECT	FE96H	CLR	FEA6H
	FE87H	CSTOP	FE97H	LIMIT	FEA7H
POSITION	FE88H	FAST	FE98H	KLIST	FEA8H
PATTERN	FE89H	REW	FE99H	ASK	FEA9H
HCOPY	FE8AH	APSS	FE9AH	KBUF	FEAAH
PLAY	FE8BH	TVPW	FE9BH	CLICK	FEABH
SOUND	FE8CH	CHANNEL	FE9CH	BOOT	FEACH
BEEP	FE8DH	VOL	FE9DH	DEVI$	FEADH
PRW	FE8EH	CRT	FE9EH	DEVO$	FEAEH
PALET	FE8FH	SCROLL	FE9FH		

===== TOKEN TABLE (3) =====

KEY WORD	TOKEN	KEY WORD	TOKEN	KEY WORD	TOKEN
INT	FF80H	SUM	FF9BH	STRPTR	FFB6H
ABS	FF81H	FRE	FF9CH	DTL	FFB7H
SIN	FF82H	LPOS	FF9DH		FFB8H
COS	FF83H	STICK	FF9EH		FFB9H
TAN	FF84H	STRIG	FF9FH	LEFT$	FFBAH
LOG	FF85H	CHR$	FFA0H	RIGHT$	FFBBH
EXP	FF86H	STR$	FFA1H	MID$	FFBCH
SQR	FF87H	HEX$	FFA2H	INKEY$	FFBDH
RND	FF88H	OCT$	FFA3H	INSTR	FFBEH
PEEK	FF89H	BIN$	FFA4H	HEXCHR$	FFBFH
ATN	FF8AH	MKI$	FFA5H	MEM$	FFC0H
SGN	FF8BH	MKS$	FFA6H	SCRN$	FFC1H
FRAC	FF8CH	MKD$	FFA7H	VARPTR	FFC2H
FIX	FF8DH	SPACE$	FFA8H	STRING$	FFC3H
PAI	FF8EH	CGPAT$	FFA9H	TIME	FFC4H
RAD	FF8FH	KANJI$	FFAAH	DAY$	FFC5H
INP	FF90H	ASC	FFABH	DATE$	FFC6H
CDBL	FF91H	LEN	FFACH	FN	FFC7H
CSNG	FF92H	VAL	FFADH	USR	FFC8H
CINT	FF93H	CVS	FFAEH	CALC	FFC9H
	FF94H	CVD	FFAFH		FFCAH
EOF	FF95H	CVI	FFB0H	ATTR$	FFCBH
FPOS	FF96H	DEVF	FFB1H	POINT	FFCCH
LOC	FF97H		FFB2H	CHARACTER$	FFCDH
LOF	FF98H	ERR	FFB3H	CMT	FFCEH
POS	FF99H	ERL	FFB4H	MIRROR$	FFCFH
FAC	FF9AH	CSRLIN	FFB5H		

参考までに、「中間コード表」をプリンタ出力するためのプログラムを掲げておきます。このプログラムは、ディスク BASIC で動くように作られています。テープ BASIC で動かしたい人は、「予約語ワードテーブル」のアドレスを以下のように訂正して下さい。

1200 行　　TAD　=&H 28 F 6
1260 行　　TAD　=&H 2 ACB
1320 行　　TAD　=&H 2 BAC

また、プリンタのない人は 1370 行～1720 行の LPRINT をすべて PRINT で置き換えて下さい。

Token Table（BASICリスト）

```
1000 ' ████████████████████████████████████████████
1010 ' █                                          █
1020 ' █              Token Table                 █
1030 ' █                                          █
1040 ' █       [ ﾁｭｳｶﾝ ｺｰﾄﾞ  ﾃｰﾌﾞﾙ ]                █
1050 ' █                                          █
1060 ' █             by Y.Shimizu                 █
1070 ' █                                          █
1080 ' █            1984.2.13 MON                 █
1090 ' █                                          █
1100 ' ████████████████████████████████████████████
1110 '
1120 WIDTH 80 : INIT : CLS 4
1130 CLICK OFF : DIM WORD$(260) : Q=0
1140 '
1150 REM ※※※ ﾁｭｳｶﾝ ｺｰﾄﾞ ﾖﾐﾀﾞｼ ※※※※※※
1160 '
1170 CFLASH 1 : PRINT "Wait a moment please !" : CFLASH 0
1180 '
1190 FLAG=1
1200 TAD =&H2921      :'<── ﾂｳｼﾞｮｳ･ﾖﾔｸｺﾞ･ﾜｰﾄﾞ･ﾃｰﾌﾞﾙ ｾﾝﾄｳ ｱﾄﾞﾚｽ
1210 TKEN=&H80        :'<── ｻｲｼｮ ﾉ ﾁｭｳｶﾝ ｺｰﾄﾞ
1220 GOSUB "TOKEN"
1230 QE1=Q-1 : NM1=Q
1240 '
1250 FLAG=2
1260 TAD =&H2AF6      :'<── ｶｸﾁｮｳ･ﾖﾔｸｺﾞ･ﾜｰﾄﾞ･ﾃｰﾌﾞﾙ ｾﾝﾄｳ ｱﾄﾞﾚｽ
1270 TKEN=&HFE80      :'<── ｻｲｼｮ ﾉ ﾁｭｳｶﾝ ｺｰﾄﾞ
1280 GOSUB "TOKEN"
1290 QE2=Q-1 : NM2=QE2-QE1
1300 '
1310 FLAG=3
```

```
1320 TAD =&H2BD7      :'<── カンスウ・ヨヤクゴ・ワード・テーブル セントウ アドレス
1330 TKEN=&HFF80      :'<── サイショ ノ チュウカン コード
1340 GOSUB "TOKEN"
1350 QE3=Q-1 : NM3=QE3-QE2
1360 '
1370 REM *** ヒョウ ヅクリ **********
1380 '
1390 CLS
1400 '
1410 FLAG=1 : GOSUB "TITLE"
1420 NM=(NM1 ¥ 3)+1
1430 FOR Q=0 TO NM-1
1440   WORD$=WORD$(Q)+"           "+WORD$(Q+NM)+"           "+WORD$(Q+NM+NM)
1450   IF Q+NM+NM>QE1 THEN WORD$=LEFT$(WORD$,56)
1460   LPRINT WORD$
1470 NEXT
1480 '
1490 LPRINT CHR$(12);
1500 '
1510 FLAG=2 : GOSUB "TITLE"
1520 NM=(NM2 ¥ 3)+1
1530 FOR Q=QE1+1 TO QE1+NM
1540   WORD$=WORD$(Q)+"           "+WORD$(Q+NM)+"           "+WORD$(Q+NM+NM)
1550   IF Q+NM+NM>QE2 THEN WORD$=LEFT$(WORD$,56)
1560   LPRINT WORD$
1570 NEXT
1580 '
1590 FOR I=1 TO 5 : LPRINT : NEXT
1600 '
1610 FLAG=3 : GOSUB "TITLE"
1620 NM=(NM3 ¥ 3)+1
1630 FOR Q=QE2+1 TO QE2+NM
1640   WORD$=WORD$(Q)+"           "+WORD$(Q+NM)+"           "+WORD$(Q+NM+NM)
1650   IF Q+NM+NM>QE3 THEN WORD$=LEFT$(WORD$,56)
1660   LPRINT WORD$
1670 NEXT
1680 '
1690 LPRINT CHR$(12);
1700 '
1710 PRINT "END !!"
1720 END
1730 '
1740 '
2000 LABEL "TITLE" :'
2010 '
2020 LPRINT USING "=====  TOKEN TABLE ( # )  =====";FLAG
2030 LPRINT : LPRINT
2040 LPRINT "KEY WORD     TOKEN           KEY WORD     TOKEN           KEY WORD     TOKEN"
2050 LPRINT "────────────────────────────────────────────────────────────────────────────"
2060 LPRINT
2070 RETURN
2080 '
3000 LABEL "TOKEN" :'
3010 '
3020 WHILE PEEK(TAD)<>&HFF
3030 '
3040   W$=""
3050   WHILE (PEEK(TAD) AND &B10000000)=0
3060     W$=W$+CHR$(PEEK(TAD))
3070     TAD=TAD+1
3080   WEND
3090   W$=W$+CHR$(PEEK(TAD) AND &B1111111) : IF ASC(LEFT$(W$,1))=0 THEN MID$(W$,1,1)=" "
3100   H$=HEAD$+RIGHT$("   "+HEX$(TKEN),4)+"H"
3110   WORD$(Q)=LEFT$(W$+"             ",13)+H$
3120   '
3130 . Q=Q+1 : TAD=TAD+1 : TKEN=TKEN+1
3140   '
3150 WEND
3160 '
3170 RETURN
3180 '
```

12—4 識別コード

前節においては、BASICの予約語がテキストエリア内に格納される様子について述べましたが、もう1つ大切なのは様々な数値データ（ GOTO の行番号など）の格納法です。予約語や変数名と区別するために、プログラム中の数値の先頭には「識別コード」とよばれるコードがつけられます。HuBASICにおける識別コードは次のようになっています。

《HuBASICにおける識別コード》

識別コード	意　　味
01H 〜 0AH	識別コードそのものが 0 〜 9 の数値を表わす。〔注1〕 （01→0、02→1、…、09→8、0A→9）
0BH	以下の2バイトは、GOTO などの後に続く行番号を16進に変換したものであることを示す。
0CH	以下の2バイトは、行番号などを、実際のメモリー上の飛び先アドレス(16進)に変換したものであることを示す。〔注2〕
0DH	以下の2バイトは、プログラム中の8進整数（&O）を16進に変換したものであることを示す。〔注3〕
0EH	以下の2バイトは、プログラム中の2進整数（&B）を16進に変換したものであることを示す。〔注3〕
0FH	以下の2バイトは、プログラム中の16進整数（&H）であることを示す。〔注3〕
12H	以下の2バイトは、10〜32767の整数を16進に変換したものであることを示す。
15H	以下の5バイトは、浮動小数点表記の単精度実数(内部表現)であることを示す。
18H	以下の8バイトは、浮動小数点表記の倍精度実数(内部表現)であることを示す。

〔注1〕00H は、1行の終わりを意味するコードとして使用するために、このようにずれてしまうのです。

〔注2〕飛び先やサブルーチンの行番号は、実行時に実際の飛び先アドレスに変換され識別コード 0CH が先頭につけられます。

〔注3〕いずれも －32768〜＋32767 の範囲の整数値です。

次に掲げるのは、識別コードの例を調べるためのサンプル・プログラムです。皆さんもいろいろ実験してみて下さい。

《識別コードを見る》

```
10 A=1
20 B=&O10
30 C=&B10
40 D=&H10
50 E%=32767
60 F!=65536!
70 G#=100000000#
80 GOTO 90
90 END

:9FC5=03 00 0A 00 41 F4 02 00 /....Aホ..
:9FCD=0A 00 14 00 42 F4 0D 08 /....Bホ..
:9FD5=00 00 0A 00 1E 00 43 F4 /......Cホ
:9FDD=0E 02 00 00 0A 00 28 00 /......(.
:9FE5=44 F4 0F 10 00 00 0B 00 /Dホ......
:9FED=32 00 45 25 F4 12 FF 7F /2.E%ホ.〒π
:9FF5=00 0E 00 3C 00 46 21 F4 /...<.F!ホ
:9FFD=15 91 00 00 00 00 00 11 /.|......
:A005=00 46 00 47 23 F4 18 9B /.F.G#ホ.┐
:A00D=3E BC 20 00 00 00 00 00 />シ .....
:A015=0A 00 50 00 80 20 0B 5A /..P._ .Z
:A01D=00 00 06 00 5A 00 98 00 /....Z.┘.
:A025=00 00 00 00 4F E5 5D 00 /....O♣].
:A02D=00 00 00 00 00 00 00 00 /........
:A035=00 00 00 00 00 00 00 00 /........
:A03D=00 00 00 00 00 00 00 00 /........
```

〔注〕モニターによるダンプリストは、テープBASICのアドレスで行なっています。ディスクBASICの方は、A8A8H番地以降をダンプして下さい。

12—5 予約語ワード・テーブル

私たちがキーボードから、BASICプログラムを入力していく際、その内容は一時「キー入力バッファ」とよばれるエリアに格納されます。この段階では中間コードは用いられず、たとえば PRINT は、ASCIIコードのまま5文字分として格納されています。

さて、1行分の入力が終わると、私たちは「リターン・キー」を押しますが、この瞬間に、BASICインタプリタのシステムが働いて、キー入力バッファの内容を中間コード形式に変換して、テキストエリアに転送します。

入力文字列を予約語かどうか判定するために、インタプリタはその内部に予約語文字列の表を持っています。これを「予約語ワード・テーブル」とよびます。テープBASICの場合、次の番地から予約語ワード・テーブルが始まっています(ディスクBASICの方は、2921H番地からダンプして下さい)。

《予約語ワードテーブルを見る》

```
:28F6=47 4F 54 CF 47 4F 53 55 /GOTﾏGOSU
:28FE=C2 47 CF 52 55 CE 52 45 /ﾂGﾏRUﾎRE
:2906=54 55 52 CE 52 45 53 54 /TURﾎREST
:290E=4F 52 C5 52 45 53 55 4D /ORﾅRESUM
:2916=C5 4C 49 53 D4 4C 4C 49 /ﾅLISﾔLLI
:291E=53 D4 44 45 4C 45 54 C5 /SﾔDELETﾅ
:2926=52 45 4E 55 CD 41 55 54 /RENUﾍAUT
:292E=CF 45 44 49 D4 46 4F D2 /ﾏEDIﾔFOﾒ
:2936=4E 45 58 D4 50 52 49 4E /NEXﾔPRIN
:293E=D4 4C 50 52 49 4E D4 49 /ﾔLPRINﾔI
:2946=4E 50 55 D4 4C 49 4E 50 /NPUﾔLINP
:294E=55 D4 49 C6 44 41 54 C1 /UﾔIﾆDATﾁ
:2956=52 45 41 C4 44 49 CD 52 /REAﾄDIﾍR
:295E=45 CD 45 4E C4 53 54 4F /EﾍENﾄSTO
:2966=D0 43 4F 4E D4 43 4C D3 /ﾐCONﾔCLﾓ
:296E=43 4C 45 41 D2 4F CE 4C /CLEAﾒOﾎL
```

いかがですか？ 予約語の文字列が(中間コードの)順にASCIIコードで格納されていますね。1つの予約語の最後の文字は、MSB(ビット7) を1にしてあります。

```
G   O   T   O
47  4F  54  [4F]  →末尾なのでMSBを1にし、[CF]とする。
```

中間コードとの関係で予約語は3種類に大別されます。通常予約語は、マイクロソフト系の標準命令を中心としたもので、 GOTO～MAXFILES の各命令、 TO～NOT の語、 ＞＜～∧ の演算子からなります。拡張予約語は、X 1独自の機能をひき出すための命令を中心とした WINDOW～DEVO$ からなります。3つ目は関数予約語で、 INT～MIRROR$ からなります。通常予約語の中間コードは1バイトで表わされますが、拡張予約語の中間コードは、 FE+□ の2バイト型、関数予約語の中間コードは、 FF+□ の2バイト型をしています。

さて、以上の3種類の予約語のワード・テーブルは、次の範囲に納められています。

《予約語ワードテーブル》

予約語種別	テープ BASIC	ディスク BASIC
通常予約語	28F6H ～ 2ACAH	2921H ～ 2AF5H
拡張予約語	2ACBH ～ 2BABH	2AF6H ～ 2BD6H
関数予約語	2BACH ～ 2CDEH	2BD7H ～ 2D0CH

〔注〕各種別のワード・テーブルの最後には、エンドマークとして、コード FFH が書かれています。

たとえば、キー入力バッファに GOSUB という文字列があると、インタプリタは予約語ワード・テーブルを先頭から検索していきます。まず中間コード初期値 80 H がセットされますが、最初の予約語 GOTO とは一致しないので、中間コードを ＋1 して、次の予約語 GOSUB との比較に移ります。ここで一致が確認されますから、 GOSUB を中間コード 81 H に変換できたことになります。このような手続きを繰り返して、インタプリタは中間コード形式のテキストを作成していく訳ですね。

12—6 予約語ジャンプ・テーブル

BASIC インタプリタは、テキストエリアに中間コード形式で格納されたプログラムを逐次翻訳しては、その処理ルーチンを呼び出す形でプログラムを実行してゆきます。その際、中間コードに対応する処理ルーチンのアドレスを求めるために、インタプリタはジャンプ・テーブルを参照します。次のダンプリストは、通常予約語の処理ジャンプ・テーブルの最初の方です。(テープ BASIC の場合です。ディスク BASIC の方は、2D0DH 番地からダンプして下さい。)

《予約語ジャンプ・テーブルを見る》

```
:2CDF=56 34 4D 32 48 34 95 17 /V4M2H4ト.
:2CE7=49 31 68 27 48 33 F7 68 /I1h'H3Bh
:2CEF=F2 68 F7 2E 09 2F 2F 21 /金hB..//!
:2CF7=A7 30 ED 17 AD 19 0C 1B /ァ0▞.ュ...
:2CFF=04 1B 35 23 A6 22 E0 34 /..5#ヲ"●4
:2D07=39 2E 9F 27 9F 87 B7 15 /9.＼'＼█キ.
:2D0F=28 21 99 1F F3 1F 82 3C /(!└.木.▁<
:2D17=CA 2E DA 33 57 16 A0 21 /ハ.レ3W. !
:2D1F=32 36 5A 20 7D 31 B0 31 /26Z }1-1
:2D27=22 32 01 31 5A 20 5A 20 /"2.1Z Z
:2D2F=5A 20 54 22 55 22 5A 20 /Z T"U"Z
:2D37=5A 20 5A 20 74 35 7A 35 /Z Z t5z5
:2D3F=7D 35 77 35 AE 26 5A 20 /}5w5ョ&Z
:2D47=AE 6A 04 6C 55 6B DD 6A /ョj.lUKンj
:2D4F=D9 38 94 36 BC 2E 71 68 /ル8┤6シ.qh
:2D57=BD 35 8F 1B A5 1A 4F 2E /ス5/.・.O.
```

上図はテープ BASIC のものですが、2CDFH 番地と 2CE0H 番地の 2 バイトには、最初の予約語 GOTO の処理ルーチンのアドレスが上下位逆転して格納されています。この場合は、3456 H 番地からが処理ルーチンであることを意味しています。

ここで一言注意しておきますと、このジャンプ・テーブルで示されるアドレスの多くは、予約語の各**中間コード**を処理するルーチンのアドレスであるという点です。前章までに紹介したシステム・サブルーチンのアドレスでは、その番地をコールすると直接に処理がなされて返ってきましたが、中間コード処理ルーチンは、中間コード形式で格納されたテキストを解釈処理実行するものなので、ユーザーが直接コールしても、期待通りに動かないか、最悪の場合は暴走することがあります。使用に際しては、十分に確認をして下さい。

予約語ジャンプ・テーブルも、予約語の 3 種類に対応して分けて格納されています。

予約語種別	テープ BASIC	ディスク BASIC
通常予約語	2CDFH ～2D9EH	2D0DH ～2E08H
拡張予約語	2D9FH ～2DFCH	2E09H ～2E2AH
関数予約語	7698H ～7737H	7EF5H ～7F94H

前節のワード・テーブルと合わせて各予約語（の中間コード）の処理ルーチンの表を作成すると次のようになります。

テープBASIC（通常ステートメント、コマンド）

ｿｳｼﾞｮｳ ｽﾃｰﾄﾒﾝﾄ､ｺﾏﾝﾄﾞ　　　TAPE BASIC (CZ-8CB01 V1.0)

ｺﾏﾝﾄﾞ	ｱﾄﾞﾚｽ	ｺﾏﾝﾄﾞ	ｱﾄﾞﾚｽ	ｺﾏﾝﾄﾞ	ｱﾄﾞﾚｽ
GOTO	3456	WIDTH	3694	><	
GOSUB	324D	OUT	2EBC	<>	
GO	3448	SEARCH	6871	=<	
RUN	1795	WAIT	35BD	<=	
RETURN	3149	PAUSE	1B8F	=>	
RESTORE	2768	WRITE	1AA5	>=	
RESUME	3348	SWAP	2E4F	=	
LIST	68F7	ERASE	205A	>	
LLIST	68F2	ERROR	2061	<	
DELETE	2EF7	ELSE	15B7	+	
RENUM	2F09	CALL	2DFD	-	
AUTO	212F	MON	0FE2	MOD	
EDIT	30A7	LOCATE	366B	¥	
FOR	17ED	SCREEN	3BFD	/	
NEXT	19AD	KEY	3A32	*	
PRINT	1B0C	LABEL	2E39	^	
LPRINT	1B04	RANDOMIZE	2E1F		
INPUT	2335	OPTION	2272		
LINPUT	22A6	LINE	3D3C		
IF	34E0	OPEN	6CE7		
DATA	2E39	CLOSE	6C77		
READ	279F	SIZE	205A		
DIM	879F	FIELD	683D		
REM	15B7	GET	4C3F		
END	2128	PUT	4E1A		
STOP	1F99	SET	683D		
CONT	1FF3	FILES	6E38		
CLS	3C82	LFILES	6E37		
CLEAR	2ECA	DEVICE	6811		
ON	33DA	NAME	205A		
LET	1657	KILL	6844		
NEW	21A0	LSET	205A		
POKE	3632	RSET	205A		
OFF	205A	INIT	6E07		
WHILE	317D	VDIM	205A		
WEND	31B0	MAXFILES	21E3		
REPEAT	3222	TO			
UNTIL	3101	STEP			
TRON	2254	THEN			
TROFF	2255	USING			
DEFINT	3574	SUB			
DEFSNG	357A	BASE			
DEFDBL	357D	TAB			
DEFSTR	3577	SPC			
DEF	26AE	EQV			
LOAD	6AAE	IMP			
SAVE	6C04	XOR			
MERGE	6B55	OR			
CHAIN	6ADD	AND			
CONSOLE	38D9	NOT			

テープBASIC（拡張ステートメント、関数）

ｶｸﾁｮｳ ｽﾃｰﾄﾒﾝﾄ､ｺﾏﾝﾄﾞ　　　ｶﾝｽｳ　　　TAPE BASIC (CZ-8CB01 V1.0)

ｺﾏﾝﾄﾞ	ｱﾄﾞﾚｽ	ｺﾏﾝﾄﾞ	ｱﾄﾞﾚｽ	ｺﾏﾝﾄﾞ	ｱﾄﾞﾚｽ
WINDOW	3772	INT	8A03	ERL	7CB3
PSET	3B4D	ABS	89FB	CSRLIN	7C3A
PRESET	3B42	SIN	8BCC	STRPTR	7C47
COLOR	3AA5	COS	8BB2	DTL	7C4D
CIRCLE	41B2	TAN	8CC6	LEFT$	7F61
POLY	41B5	LOG	8F84	RIGHT$	7F79
PAINT	47C1	EXP	8E6C	MID$	7F97
POSITION	4F39	SQR	8A5C	INKEY$	803D
PATTERN	4F45	RND	8E3A	INSTR	818E
HCOPY	50AC	PEEK	8E31	HEXCHR$	821B
PLAY	44E7	ATN	8AEA	MEM$	807D
SOUND	475D	SGN	8DD7	SCRN$	80B4
BEEP	477A	FRAC	8A3C	VARPTR	7F07
PRW	3B23	FIX	8A33	STRING$	8135
PALET	3AE6	PAI	8E08	TIME	800C
LAYER	4AF9	RAD	8E00	DAY$	7FEC

CANVAS	4AE5	INP	8E24	DATE$	7FFD
CREV	416A	CDBL	53D9	FN	8820
CFLASH	4181	CSNG	53F6	USR	82E1
CGEN	4191	CINT	5A50	ATTR$	205A
CSIZE	41A1	DSKF	680E	POINT	810E
EJECT	4A86	EOF	6CBF	CHARACTER$	80A9
CSTOP	4A88	FPOS	680E	CMT	7F19
FAST	4A8B	LOC	680E	MIRROR$	82BA
REW	4A8E	LOF	680E		
APSS	4A98	POS	67E7		
TVPW	52DA	FAC	8A96		
CHANNEL	52F3	SUM	8A71		
VOL	52FF	FRE	7C24		
CRT	5331	LPOS	67DF		
SCROLL	4BEA	STICK	7C6F		
EFFECT	4C2A	STRIG	7C53		
GRAPH	3BFD	CHR$	7CD0		
MUSIC	44E7	STR$	7E9B		
TEMPO	44E7	HEX$	7D4A		
CURSOR	366B	OCT$	7D34		
VERIFY	69EA	BIN$	7D3F		
CLR	21EE	MKI$	7DC6		
LIMIT	2ECD	MKS$	7DCB		
KLIST	393D	MKD$	7DD0		
ASK	52A1	SPACE$	7DDC		
KBUF	4C2D	CGPAT$	7E02		
CLICK	52C8	KANJI$	7E26		
BOOT	5293	ASC	7EBE		
DEVI$	6719	LEN	7ECA		
DEVO$	676D	VAL	7ED2		
		CVS	7EEC		
		CVD	7EF0		
		CVI	7EE8		
		ERR	7C42		

ディスクBASIC（通常ステートメント、コマンド）

ﾂｳｼﾞｮｳ ｽﾃｰﾄﾒﾝﾄ､ｺﾏﾝﾄﾞ　　DISK BASIC (CZ-8FB01 V1.0)

ｺﾏﾝﾄﾞ	ｱﾄﾞﾚｽ	ｺﾏﾝﾄﾞ	ｱﾄﾞﾚｽ	ｺﾏﾝﾄﾞ	ｱﾄﾞﾚｽ
GOTO	3488	WIDTH	36C6	><	
GOSUB	327F	OUT	2EEA	<>	
GO	347A	SEARCH	613D	=<	
RUN	17B0	WAIT	35EF	<=	
RETURN	317B	PAUSE	1BAA	=>	
RESTORE	2793	WRITE	1AC0	>=	
RESUME	337A	SWAP	2E7D	=	
LIST	61C3	ERASE	206C	>	
LLIST	61BE	ERROR	2073	<	
DELETE	2F25	ELSE	15C9	+	
RENUM	2F37	CALL	2E2B	-	
AUTO	2150	MON	0FE2	MOD	
EDIT	30D5	LOCATE	369D	¥	
FOR	1808	SCREEN	3C2F	/	
NEXT	19C8	KEY	3A64	*	
PRINT	1B27	LABEL	2E67	^	
LPRINT	1B1F	RANDOMIZE	2E4D		
INPUT	2359	OPTION	2296		
LINPUT	22CA	LINE	3D6E		
IF	3512	OPEN	65BC		
DATA	2E67	CLOSE	6547		
READ	27CA	SIZE	206C		
DIM	906A	FIELD	5E69		
REM	15C9	GET	4C76		
END	2149	PUT	4E57		
STOP	1FB4	SET	6093		
CONT	2005	FILES	6719		
CLS	3CB4	LFILES	6718		
CLEAR	2EF8	DEVICE	5E3D		
ON	340C	NAME	679C		
LET	166A	KILL	6110		
NEW	21C4	LSET	675C		
POKE	3664	RSET	672F		
OFF	206C	INIT	66E8		
WHILE	31AF	VDIM	206C		
WEND	31E2	MAXFILES	2207		
REPEAT	3254	TO			
UNTIL	3133	STEP			
TRON	2278	THEN			
TROFF	2279	USING			
DEFINT	35A6	SUB			
DEFSNG	35AC	BASE			
DEFDBL	35AF	TAB			
DEFSTR	35A9	SPC			
DEF	26D9	EQV			
LOAD	637A	IMP			
SAVE	64D4	XOR			
MERGE	6423	OR			
CHAIN	63A9	AND			
CONSOLE	390B	NOT			

ディスクBASIC（拡張ステートメント、関数）

ｶｸﾁｮｳ ｽﾃｰﾄﾒﾝﾄ､ｺﾏﾝﾄﾞ

ｺﾏﾝﾄﾞ	ｱﾄﾞﾚｽ
WINDOW	37A4
PSET	3B7F
PRESET	3B74
COLOR	3AD7
CIRCLE	41E5
POLY	41E8
PAINT	47F4
POSITION	4F76
PATTERN	4F82
HCOPY	50E9
PLAY	451A
SOUND	4790
BEEP	47AD
PRW	3B55
PALET	3B18
LAYER	4B30
CANVAS	4B1C
CREV	419D
CFLASH	41B4
CGEN	41C4
CSIZE	41D4
EJECT	4AB9
CSTOP	4ABB
FAST	4ABE
REW	4AC1
APSS	4ACB
TVPW	531A
CHANNEL	5333
VOL	533F
CRT	5371
SCROLL	4C21
EFFECT	4C61
GRAPH	3C2F
MUSIC	451A
TEMPO	451A
CURSOR	369D
VERIFY	62B6
CLR	2212
LIMIT	2EFB
KLIST	396F
ASK	52DE
KBUF	4C64
CLICK	530B
BOOT	52D0
DEVI$	5CBB
DEVO$	5D0F

ｶﾝｽｳ

ｺﾏﾝﾄﾞ	ｱﾄﾞﾚｽ
INT	92E6
ABS	92DE
SIN	94AF
COS	9495
TAN	95A9
LOG	9867
EXP	974F
SQR	933F
RND	971D
PEEK	9714
ATN	93CD
SGN	96BA
FRAC	931F
FIX	9316
PAI	96EB
RAD	96E3
INP	9707
CDBL	5419
CSNG	5436
CINT	5A90
EOF	658F
FPOS	5DFE
LOC	5E18
LOF	5E23
POS	5D8B
FAC	9379
SUM	9354
FRE	8481
LPOS	5D83
STICK	84CC
STRIG	84B0
CHR$	852D
STR$	86F8
HEX$	85A7
OCT$	8591
BIN$	859C
MKI$	8623
MKS$	8628
MKD$	862D
SPACE$	8639
CGPAT$	865F
KANJI$	8683
ASC	871B
LEN	8727
VAL	872F
CVS	8749
CVD	874D
CVI	8745
DEVF	5DB2
ERR	849F

DISK BASIC (CZ-8FB01 V1.0)

ｺﾏﾝﾄﾞ	ｱﾄﾞﾚｽ
ERL	8510
CSRLIN	8497
STRPTR	84A4
DTL	84AA
LEFT$	87E0
RIGHT$	87F8
MID$	8816
INKEY$	88BC
INSTR	8A0D
HEXCHR$	8A9A
MEM$	88FC
SCRN$	8933
VARPTR	8764
STRING$	89B4
TIME	888B
DAY$	886B
DATE$	887C
FN	90EB
USR	8B60
CALC	8BB5
ATTR$	602B
POINT	898D
CHARACTER$	8928
CMT	8798
MIRROR$	8B39

予約語ワード・テーブルとジャンプ・テーブルの検索手続きの参考までに、以上の表をプリンタ出力するために用いたプログラムを掲げておきます。

BASIC予約語ワード・テーブル作成ソフト（BASICプログラム）

```
1000 ' ▀▀▀▀▀▀▀▀▀▀▀▀▀▀▀▀▀▀▀▀▀▀▀▀▀▀▀▀▀▀▀▀▀▀▀▀
1010 ' █                                  █
1020 ' █  BASIC ﾖﾔｸｺﾞ ﾜｰﾄﾞ･ﾃｰﾌﾞﾙ ｻｸｾｲ ｿﾌﾄ  █
1030 ' █                                  █
1040 ' █      by Yasuhiro Shimizu         █
1050 ' █      1984.1.15 SUN               █
1060 ' █                                  █
1070 ' ▄▄▄▄▄▄▄▄▄▄▄▄▄▄▄▄▄▄▄▄▄▄▄▄▄▄▄▄▄▄▄▄▄▄▄▄
1080 '
1090 CLEAR &HFF00
1100 WIDTH 40 :INIT
1110 DIM WADR(3),JADR(3),COM$(3,50)
1120 '
1130 ' ■■■ BASIC ﾉ ｾﾝﾀｸ ■■■
1140 '
1150 PRINT "BASIC ﾊ ﾅﾆﾃﾞｽｶ ?  ﾊﾞﾝｺﾞｳ ﾃﾞ ｴﾗﾝﾃﾞ ｸﾀﾞｻｲ｡"
1160 PRINT
1170 PRINT " 1: ﾃｰﾌﾟ  BASIC  (CZ-8CB01)"
1180 PRINT " 2: ﾃﾞｨｽｸ BASIC  (CZ-8FB01)"
1190 PRINT
1200 I$=INKEY$ :IF I$="" THEN 1200
1210 IF I$<"1" OR I$>"2" THEN 1200
1220 I=VAL(I$)
1230 '
```

```
1240 ' ■■■ ｱﾄﾞﾚｽ ﾉ ｾｯﾃｲ ■■■
1250 '
1260 ON I GOTO 1300,1420
1270 '
1280 ' ── ﾃｰﾌﾟ BASIC ──
1290 '
1300 WADR(1)=&H28F6 :'<── ｿｳｼﾞｮｳ ﾖﾔｸｺﾞ ﾜｰﾄﾞ ﾃｰﾌﾞﾙ
1310 WADR(2)=&H2ACB :'<── ｶｸﾁｮｳ  ﾖﾔｸｺﾞ ﾜｰﾄﾞ ﾃｰﾌﾞﾙ
1320 WADR(3)=&H2BAC :'<── ｶﾝｽｳ   ﾖﾔｸｺﾞ ﾜｰﾄﾞ ﾃｰﾌﾞﾙ
1330 JADR(1)=&H2CDF :'<── ｿｳｼﾞｮｳ ﾖﾔｸｺﾞ ｴﾝﾄﾘｰ ｼﾞｬﾝﾌﾟ ﾃｰﾌﾞﾙ
1340 JADR(2)=&H2D9F :'<── ｶｸﾁｮｳ  ﾖﾔｸｺﾞ ｴﾝﾄﾘｰ ｼﾞｬﾝﾌﾟ ﾃｰﾌﾞﾙ
1350 JADR(3)=&H7698 :'<── ｶﾝｽｳ   ﾖﾔｸｺﾞ ｴﾝﾄﾘｰ ｼﾞｬﾝﾌﾟ ﾃｰﾌﾞﾙ
1355    EADR=&H2A85  :'<── ｼｮﾘ ｱﾄﾞﾚｽ ｦ ﾓﾂ ｿｳｼﾞｮｳ ﾖﾔｸｺﾞ ﾜｰﾄﾞ ﾃｰﾌﾞﾙ
1360 BAS$="TAPE BASIC (CZ-8CB01 V1.0)"
1370 '
1380 GOTO 1530
1390 '
1400 ' ── ﾃﾞｨｽｸ BASIC ──
1410 '
1420 WADR(1)=&H2921
1430 WADR(2)=&H2AF6
1440 WADR(3)=&H2BD7
1450 JADR(1)=&H2D0D
1460 JADR(2)=&H2DCD
1470 JADR(3)=&H7EF5
1475    EADR=&H2AB0
1480 BAS$="DISK BASIC (CZ-8FB01 V1.0)"
1490 '
1500 ' ■■■ ﾒｲﾝ ﾙｰﾁﾝ ■■■
1510 '
1520 WIDTH 80
1530 FOR I=1 TO 3 :'◆◆◆◆◆◆◆◆◆◆◆◆◆◆◆◆◆◆◆◆◆◆◆◆◆◆◆◆◆◆◆◆◆◆◆◆◆◆◆◆◆◆◆◆◆◆◆◆◆◆◆◆◆◆◆◆◆◆◆◆
1540 '
1550 LABEL "ｼｮｷ ｾｯﾃｲ" :'■■■■■■■■■■
1560 '
1570 WAD=WADR(I) :JAD=JADR(I) :IF I=1 THEN EAD=EADR ELSE EAD=&H7FFF
1580 FOR N=1 TO 3
1590   FOR L=1 TO 50
1600     COM$(N,L)=""
1610   NEXT L
1620 NEXT N
1630 PAGE=1 :FLAG=0 :OVER=0
1640 '
1650 ON I GOTO 1660,1670,1680
1660   M$="ｿｳｼﾞｮｳ ｽﾃｰﾄﾒﾝﾄ､ｺﾏﾝﾄﾞ" :GOTO "1 ﾍﾟｰｼﾞ ﾌﾞﾝ"
1670   M$="ｶｸﾁｮｳ ｽﾃｰﾄﾒﾝﾄ､ｺﾏﾝﾄﾞ " :GOTO "1 ﾍﾟｰｼﾞ ﾌﾞﾝ"
1680   M$="ｶﾝｽｳ              "
1690 '
1700 LABEL "1 ﾍﾟｰｼﾞ ﾌﾞﾝ" :'■■■■■■■■■■
1710 '
1720 LPRINT M$+"             "+BAS$+"                    page";I
1730 LPRINT :LPRINT
1740 CFLASH 1 :PRINT "ｽｺｼ ｵﾏﾁ ｸﾀﾞｻｲ｡" :CFLASH 0
1750 '
1760 N=1 :FLAG=0 :GOSUB "N ﾚﾂ ｼｮﾘ"
1770 IF OVER=1 THEN 1820
1780 N=2 :FLAG=0 :GOSUB "N ﾚﾂ ｼｮﾘ"
1790 IF OVER=1 THEN 1820
1800 N=3 :FLAG=0 :GOSUB "N ﾚﾂ ｼｮﾘ"
1810 '
1820 PRINT CHR$(&H1E,&H1E,&H1A)
1830 LPRINT "ｺﾏﾝﾄﾞ       ｱﾄﾞﾚｽ           ｺﾏﾝﾄﾞ        ｱﾄﾞﾚｽ            ｺﾏﾝﾄﾞ         ｱﾄﾞﾚｽ"
1840 LPRINT :LPRINT
1850 '
1860 LI=1 :'<─ ﾗｲﾝ･ｶｳﾝﾀｰ ｼｮｷｶ
1870 LPRINT COM$(1,LI)+"           "+COM$(2,LI)+"             "+COM$(3,LI)
1880 LI=LI+1 :IF LI<51 THEN 1870
1890 IF LI=51 AND OVER=0 THEN PAGE=PAGE+1 :LPRINT CHR$(12) :GOTO "1 ﾍﾟｰｼﾞ ﾌﾞﾝ"
1900 '
1910 LPRINT CHR$(12)
1920 '
1930 NEXT I :'◆◆◆◆◆◆◆◆◆◆◆◆◆◆◆◆◆◆◆◆◆◆◆◆◆◆◆◆◆◆◆◆◆◆◆◆◆◆◆◆◆◆◆◆◆◆◆◆◆◆◆◆◆◆◆◆◆◆◆◆◆◆
1940 '
1950 PRINT :PRINT
1960 PRINT "ｻｸｾｲ ｼｭｳﾘｮｳ" :END
1970 '
1980 '
1990 ' ■■■ ｻﾌﾞﾙｰﾁﾝ ■■■
2000 '
2010 LABEL "N ﾚﾂ ｼｮﾘ" :'■■■■■■■■■■
2020 '
2030 FOR L=1 TO 50
2040   '
2050   LABEL "ｶｲﾄﾞｸ" :'────
2060   W=PEEK(WAD)
2070   IF W=&HFF THEN "ｼｭｳﾘｮｳ"                            :'<─ ｴﾝﾄﾞ･ﾏｰｸ
2080   IF W=&H80 THEN "ﾐﾃｲｷﾞ"                             :'<─ "_"(=&H80) ﾉ ﾄｷ
2090   IF (W AND &H80)=&H80 THEN FLAG=1 :W=(W AND &H7F) :'<─ ｺﾏﾝﾄﾞ ﾉ ｵﾜﾘ
2100   '
2110   COM$(N,L)=COM$(N,L)+CHR$(W)
2120   WAD=WAD+1
2130   IF FLAG=0 THEN "ｶｲﾄﾞｸ"
2140   '
2150   C$=LEFT$(COM$(N,L)+"             ",13)
2160   COM$(N,L)=C$ :'<── ﾉｺﾘ ｦ ｽﾍﾟｰｽ ﾃﾞ ｳﾒﾙ!
2170   IF WAD<EAD THEN GOSUB "ｱﾄﾞﾚｽ" ELSE AD$="
2180   COM$(N,L)=COM$(N,L)+AD$
2190   JAD=JAD+2
2200   '
2210   FLAG=0
2220   GOTO "ﾂｷﾞ ﾉ ｷﾞｮｳ"
2230   '
```

```
2240    LABEL "ﾐﾃｲｷﾞ" :'────
2250    WAD=WAD+1
2260    JAD=JAD+2
2270    GOTO "ｶｲﾄﾞｸ"
2280    '
2290    LABEL "ｼｭｳﾘｮｳ" :'────
2300    OVER=1
2310    COM$(N,L)=""
2320    '
2330 LABEL "ﾂｷﾞ ﾉ ｷﾞｮｳ" :'────
2340 NEXT L
2350 '
2360 RETURN
2370 '
2380 '
2390 LABEL "ｱﾄﾞﾚｽ" :'
2400 '
2410 AL$=RIGHT$("0"+HEX$(PEEK(JAD)),2)
2420 AH$=RIGHT$("0"+HEX$(PEEK(JAD+1)),2)
2430 AD$=AH$+AL$
2440 RETURN
2450 '
2460 '
2470 '
```

12－7　数値の内部表現

BASIC プログラムの中で、私たちは様々な数値を扱いますが、 PRINT 文などによりディスプレイやプリンタに出力される数値の表示（**外部表現**という）と、その数値が実際にメモリー内に格納されている形式（**内部表現**という）とは大変異なっているという点を注意する必要があります。本節では、数値データの内部表現の問題を説明します。

HuBASIC で扱われる数値は、次の３つに大別されます。

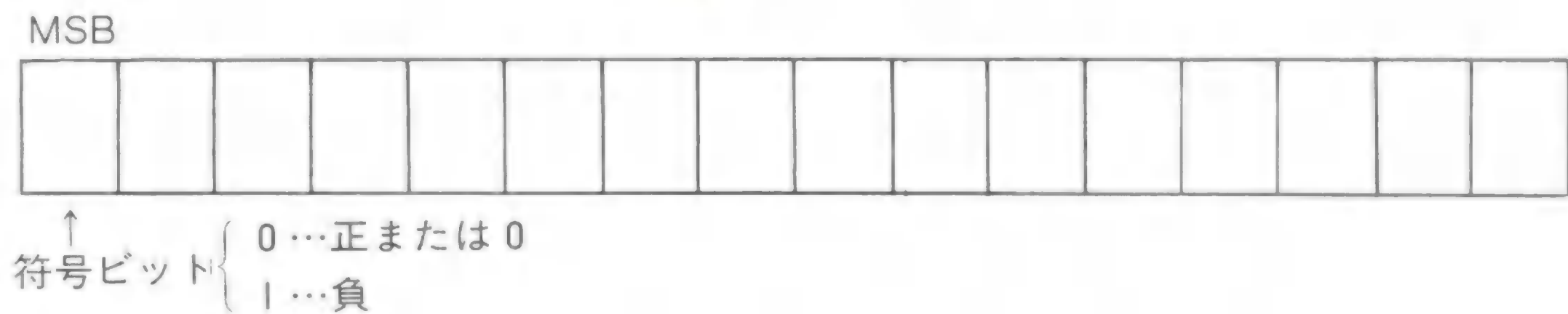
- 整数
- 単精度実数
- 倍精度実数

このうち、整数と実数とでは内部表現が大変に異なります。

整数は、HuBASIC では２バイト（16 ビット）のデータとして扱われます。

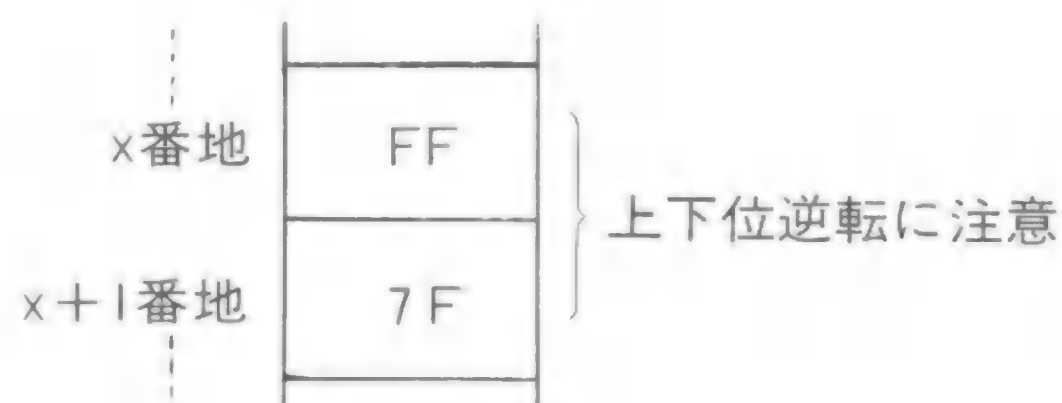
《**整数データ**》

MSB															

↑
符号ビット｛0…正または0／1…負

負の整数は、「２の補数」とよばれる方法で扱われるために、MSB が１の場合は負の数になることに注意しましょう。従って、整数の範囲としては、次図のようになります。

2　進　表　示	16 進 表 示	10 進 表 示
0 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1	7 F F F	＋32767
0 1 1 1 1 1 1 1 1 1 1 1 1 1 1 0	7 F F E	＋32766
⋮	⋮	⋮
0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 1	0 0 0 1	＋1
0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0	0 0 0 0	0
1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1	F F F F	－1
1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 0	F F F E	－2
⋮	⋮	⋮
1 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0	8 0 0 0	－32768

メモリーには、16 ビットの上位 8 ビットと下位 8 ビットが逆転して格納されます。ですから、 +32767 という整数値の内部表現は

x番地	FF	上下位逆転に注意
x+1番地	7F	

となる訳ですね。

では次に、実数の扱いに進みます。実数は**「浮動小数点形式」**あるいは**「指数形式」**で扱われます。たとえば、 124.6 のような小数点つきの数を扱うとき、小数点の位置を常に固定して表示する方法を**「固定小数点形式」**とよびます。私たちが普段見慣れているのは、多くこの形です。これに対して、小数点の位置を適宜移動させて、 1.246×10^2 のように表示することもできます。この形を**「浮動小数点形式」**とよぶのです。理工系の方々には、なじみのある表現方法ですね（たとえば光の速度 2.9979×10^8m ／秒 など）。

もう 1 つ注意することは、（小数点つきの）実数も 2 進数として扱われる点です。10 進小数で、 0.123 というのは、

$$0.123=\frac{1}{10}+\frac{2}{10^2}+\frac{3}{10^3}\ \text{(10進数)}$$

という意味ですね。分母の 10, 10^2, 10^3 は 10 進数での「小数点以下の各位」に対応しています。 2 進小数でも考え方は同じです。たとえば、 2 進小数で 0.1011 というのは、

$$0.1011(\text{2 進})=\frac{1}{2}+\frac{0}{2^2}+\frac{1}{2^3}+\frac{1}{2^4}=0.6875(\text{10進})$$

という意味があります。

さて、HuBASIC では、2 進小数を次のような浮動小数点形式で扱うことに決めています（0 以外の数は必ずこの形にできます）。

$\pm 1.a_1a_2a_3\cdots\cdots a_n\cdots\cdots\times2^k$ （kは指数。a_jたちは 0 か 1 ）

次に例を挙げますから、皆さんも考えてみて下さい。

〔例 1〕 +1.5 （10 進）＝ +1.1 （2 進）$\times2^0$
〔例 2〕 −10 （10 進）＝ −1.01 （2 進）$\times2^3$
〔例 3〕 +0.6875 （10 進）＝ +1.011 （2 進）$\times2^{-1}$

このような浮動小数点形式で表示された実数データから、次のように内部表現を作ります。

①指数部…指数kと81Hとを加える。ただし、もともと数値が0のときは、00Hとする。
②仮数部…仮数1.$a_1a_2a_3$…では先頭が必ず1なので、このビットを符号ビット（正のとき0、負のとき1）として使用し、左から必要なバイト数だけとる。単精度では4バイト、倍精度では7バイトを採用する。末尾を丸めなくてはならない場合は、次の桁を「0捨1入」する。
③メモリー内には指数部1バイト、仮数部（4または7バイト）の順に格納する。

早速、上記の3つの例を内部表現に直してみましょう（単精度の場合を考えます)。

〔例1の内部表現〕

指数部　81 H+0=81 H
仮数部　01000000……00000000　=　40000000 H
↑
符号
従って、　81 H,40 H,00 H,00 H,00 H　の順に格納されます。

〔例2の内部表現〕

指数部　81 H+3=84 H
仮数部　10100000……00000000　=　A 0000000 H
↑
符号
従って、　84 H,A 0 H,00 H,00 H,00 H　の順に格納されます。

〔例3の内部表現〕

指数部　81 H−1=80 H
仮数部　00110000……00000000　=　30000000 H
↑
符号
従って、　80 H,30 H,00 H,00 H,00 H　の順に格納されます。

上記の3例では、いずれも2進有限小数を扱いましたが、たとえば10進での 0.1 は2進小数としては無限小数となります。最後にこの例を考えておきましょう。

〔10進 0.1 の内部表現（単精度)〕

0.1（10進）＝ 1.1001100110011001……(2進）$\times 2^{-4}$（無限循環小数）

指数部　　81 H－4＝7 DH

仮数部　　01001100110011001100110011001101 ＝ 4 CCCCCCDH

↑（符号）　　　↑（次桁を0捨1入した）

従って、 7 DH,4 CH,CCH,CCH,CDH の順に格納されます。

数値の内部表現を調べるプログラムを掲げておきますので、いろいろな数値を入力して、観察して下さい。本プログラムでは、内部表現8バイトを出力しますが、単精度対応なので5バイト分のみ有効です。倍精度を出力したい人は、340行と380行の変数Aを A# に、変数BをB#に変えて下さい。

内部表現（BASIC＋マシン語）

```
100 REM ************************
110 REM *                      *
120 REM *    ﾅｲﾌﾞ ﾋｮｳｹﾞﾝ      *
130 REM *                      *
140 REM *                      *
150 REM *   by Y.Shimizu       *
160 REM *                      *
170 REM *   1984.4.28 SAT      *
180 REM *                      *
190 REM ************************
200 '
210 '----------
220 ' ﾏｴ ｼｮﾘ
230 '----------
240 '
250 CLEAR &HF100
260 CLS
270 GOSUB "ｶｷｺﾐ"
280 DEFUSR=&HFE00
290 '
300 '----------
310 ' ﾒｲﾝ ﾙｰﾁﾝ
320 '----------
330 '
340 BEEP :INPUT "ｽｳﾁ = ";A
350 PRINT
360 PRINT "ﾅｲﾌﾞ ﾋｮｳｹﾞﾝ"
370 PRINT "―――――――――――"
380 B=USR(A)
390 PRINT :PRINT
400 GOTO 340
410 '
420 '
430 '----------
440 ' ｻﾌﾞ ﾙｰﾁﾝ
450 '----------
460 '
470 LABEL "ｶｷｺﾐ" :'========
480 '
490 ADR=&HFE00 :RESTORE 590
500 FOR I=0 TO 16
510   READ M$ :POKE ADR+I,VAL("&H"+M$)
520 NEXT I
530 RETURN
540 '
550 '------------
560 ' ﾏｼﾝｺﾞ ﾃﾞｰﾀ
570 '------------
580 '
590 DATA 06,08            :'LD B,8
600 DATA C5               :'PUSH BC
610 DATA E5               :'PUSH HL
620 DATA 7E               :'LD A,(HL)
630 DATA CD,07,12         :'CALL 1207H
640 DATA CD,A7,04         :'CALL 04A7H
650 DATA E1               :'POP HL
660 DATA 23               :'INC HL
670 DATA C1               :'POP BC
680 DATA 10,F2            :'DJNZ LOOP
690 DATA C9               :'RET
700 '
```

12−8　数値変数の格納形式

HuBASIC で扱われる変数は次の４種に大別されます。

- 数値変数………
  - 整数型………………A％など
  - 単精度実数型………B！など
  - 倍精度実数型………C♯など
- 文字列変数………………………………D＄など

このうち本節では、数値変数の格納形式について述べます。

変数たちを格納する**変数エリア**は、テキストエリアのすぐ後に置かれます。BASIC がコールド・スタート（14A0H 番地をコールすると、メモリー内がクリアされて、BASIC がスタートします）した直後に、プログラムを入力し、テキストエリアをダンプしても変数エリアは見えません。プログラムを RUN して初めて、インタプリタにより設けられるエリアです。では早速、観察してみましょう。次のプログラムを入力し、RUN させた後にテキストエリアをダンプして下さい（ディスク BASIC の方は、A8A8H 番地以降)。

《変数エリアを見る》

```
10 A%=16
20 B!=.6875
30 C#=.1#
40 END

:9FC5=0B 00 0A 00 41 25 F4 12 /....A%ж.
:9FCD=10 00 00 0E 00 14 00 42 /.......B
:9FD5=21 F4 15 80 30 00 00 00 /!ж._0...
:9FDD=00 11 00 1E 00 43 23 F4 /.....C#ж
:9FE5=18 7D 4C CC CC CC CC CC /.)Lﾌﾌﾌﾌﾌ
:9FED=CD 00 06 00 28 00 98 00 /ﾍ...(.ｰ.
:9FF5=00 00 02 01 41 10 00 05 /....A...
:9FFD=01 42 80 30 00 00 00 08 /.B_0....
:A005=01 43 7D 4C CC CC CC CC /.C)Lﾌﾌﾌﾌ
:A00D=CC CD 00 00 4F E5 5D 00 /ﾌﾍ..O♣].
:A015=00 00 00 00 00 00 00 00 /........
:A01D=00 00 00 00 00 00 00 00 /........
:A025=00 00 00 00 00 00 00 00 /........
:A02D=00 00 00 00 00 00 00 00 /........
:A035=00 00 00 00 00 00 00 00 /........
:A03D=00 00 00 00 00 00 00 00 /........
```

ここで、9FF7H 番地以降が変数エリアです。データの読み方を以下に書きます。

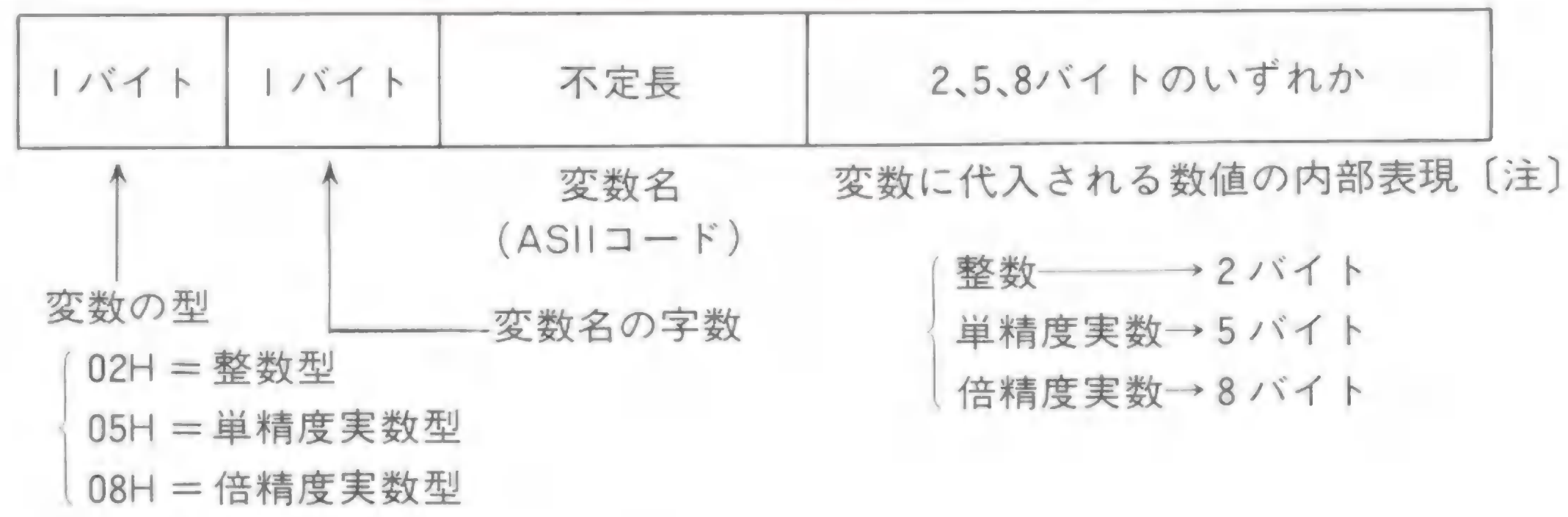
アドレス	コード	意　味	アドレス	コース	意　味	アドレス	コード	意　味
9FF7	02	変数の型(整数)	A001	00	(５バイト)	A00B	CC	(８バイト)
9FF8	01	変数名の字数	A002	00		A00C	CC	
9FF9	41	変数名A	A003	00		A00D	CC	
9FFA	10	数値16	A004	08	変数の型(倍精度)	A00E	CD	
9FFB	00		A005	01	変数名の字数	A00F	00	変数エンド・コード
9FFC	05	変数の型(単精度)	A006	43	変数名C	A010	00	↓これ以降は
9FFD	01	変数名の字数	A007	7D	数値0.1	A011	4F	ファイル用
9FFE	42	変数名B	A008	4C	の倍精度内部	A012	E5	ストリング
9FFF	80	数値0.6875	A009	CC	表現	A013	5F	バッファ
A000	30	の内部表現	A00A	CC				

このように、変数は登場する順に変数エリアに格納され、各変数の内容が一定の形式で管理されます。ここに登場する変数は配列ではない**単純変数**とよばれるものですが、各単純変数の格納形式をまとめると以下のようになります。

《単純数値変数の格納形式》

1バイト	1バイト	不定長	2、5、8バイトのいずれか
↑ 変数の型	↑ 変数名の字数	変数名（ASIIコード）	変数に代入される数値の内部表現〔注〕

変数の型
- 02H ＝整数型
- 05H ＝単精度実数型
- 08H ＝倍精度実数型

- 整数 → 2バイト
- 単精度実数 → 5バイト
- 倍精度実数 → 8バイト

〔注〕BASIC関数の VARPTR は、各変数のデータ部分先頭アドレスを返すものです。

次に配列変数について考えます。次のサンプルプログラムで変数エリアを観察します。

《配列変数を観察する》

```
10 DIM A%(3)
20 FOR I=0 TO 2
30    A%(I)=I
40 NEXT
50 END

:9FC5=0C 00 0A 00 96 20 41 25 /....+ A%
:9FCD=28 04 29 00 0E 00 14 00 /(.).....
:9FD5=8D 20 49 F4 01 20 E0 20 /█ I*. ●
:9FDD=03 00 0E 00 1E 00 20 20 /......
:9FE5=41 25 28 49 29 F4 49 00 /A%(I)*I.
:9FED=06 00 28 00 8E 00 06 00 /..(.█...
:9FF5=32 00 98 00 00 00 82 0F /2.┘...▬
:9FFD=00 01 41 01 04 00 00 00 /..A.....
:A005=01 00 02 00 00 00 05 01 /........
:A00D=49 82 40 00 00 00 00 00 /I▂a.....
:A015=4F E5 5D 00 00 00 00 00 /O♣].....
:A01D=00 00 00 00 00 00 00 00 /........
:A025=00 00 00 00 00 00 00 00 /........
:A02D=00 00 00 00 00 00 00 00 /........
:A035=00 00 00 00 00 00 00 00 /........
:A03D=00 00 00 00 00 00 00 00 /........
```

《配列変数の格納状況》

アドレス	コード	意味
9FFB	82	変数の型（整数）
9FFC	0F	以下のバイト数（15バイト）
9FFD	00	
9FFE	01	変数名の字数
9FFF	41	変数名A
A000	01	配列の次元
A001	04	配列の大きさ（0〜3の4個）
A002	00	
A003	00	A%(0)の内容
A004	00	

アドレス	コード	意味
A005	01	A%(1)の内容
A006	00	
A007	02	A%(2)の内容
A008	00	
A009	00	A%(3)の内容
A00A	00	
A00B	05	変数Iの型（単精度）
A00C	01	変数名の字数
A00D	49	変数名I

アドレス	コード	意味
A00E	82	変数Iの内容（3の単精度内部表現）
A00F	40	
A010	00	
A011	00	
A012	00	
A013	00	変数エンドコード
A014	00	↓ファイル用ストリングバッファ
A015	4F	

〔注〕HuBASIC では、起動直後の数値変数は単精度に設定されるので、変数 I は単精度として扱われています。

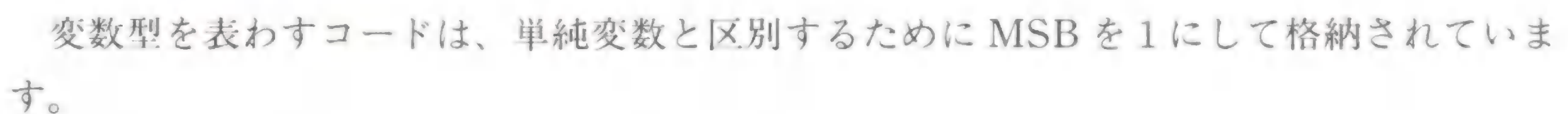
9 FFBH～A 00 AH 番地が、配列変数 A %(0)～A %(3) にあてられています。1組の配列変数の格納形式をまとめると、以下のようになります。

《配列数値変数の格納形式》

1バイト	2バイト	1バイト	不定長	1バイト	2バイト	2、5、8バイト	2、5、8バイト	……
変数の型 82H＝整数型 85H＝単精度実数型 88H＝倍精度実数型	以下のバイト数	変数名の字数	変数名	配列の次元	配列の大きさ	0番目の内容（内部表現）	1番目の内容（内部表現）	

変数型を表わすコードは、単純変数と区別するためにMSBを1にして格納されています。

12－9 文字列変数の格納形式

今度は文字列変数について考察します。前節と同様に次のサンプルプログラムを RUN して、変数エリアの状況を見てみましょう。

サンプルプログラム

```
10 A$="123"
20 B$="ABCD"
30 END
```

《変数エリア》

アドレス	コード	意味	アドレス	コード	意味	アドレス	コード	意味
9FE8	03	変数の型(文字)	9FED	02	(3バイト)	9FF2	27	ディスクリプタ
9FE9	01	変数名の字数	9FEE	03	変数の型(文字)	9FF3	02	(3バイト)
9FEA	41	変数名A	9FEF	01	変数名の字数	9FF4	00	変数エンドコード
9FEB	03	ストリング	9FF0	42	変数名B	9FF5	00	ファイル用
9FEC	22	ディスクリプタ	9FF1	04	ストリング	9FF6	4F	ストリングバッファ

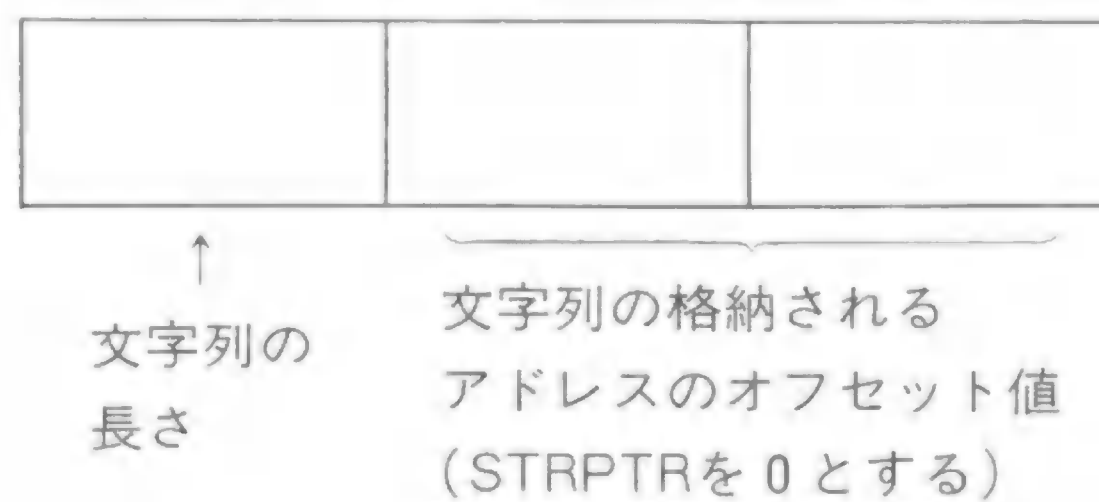
これより、次のようにまとめられます。

《単純文字列変数の格納形式》

1バイト	1バイト	不定長	3バイト
変数型 03H＝文字列型	変数名の字数	変数名	ストリング・ディスクリプタ

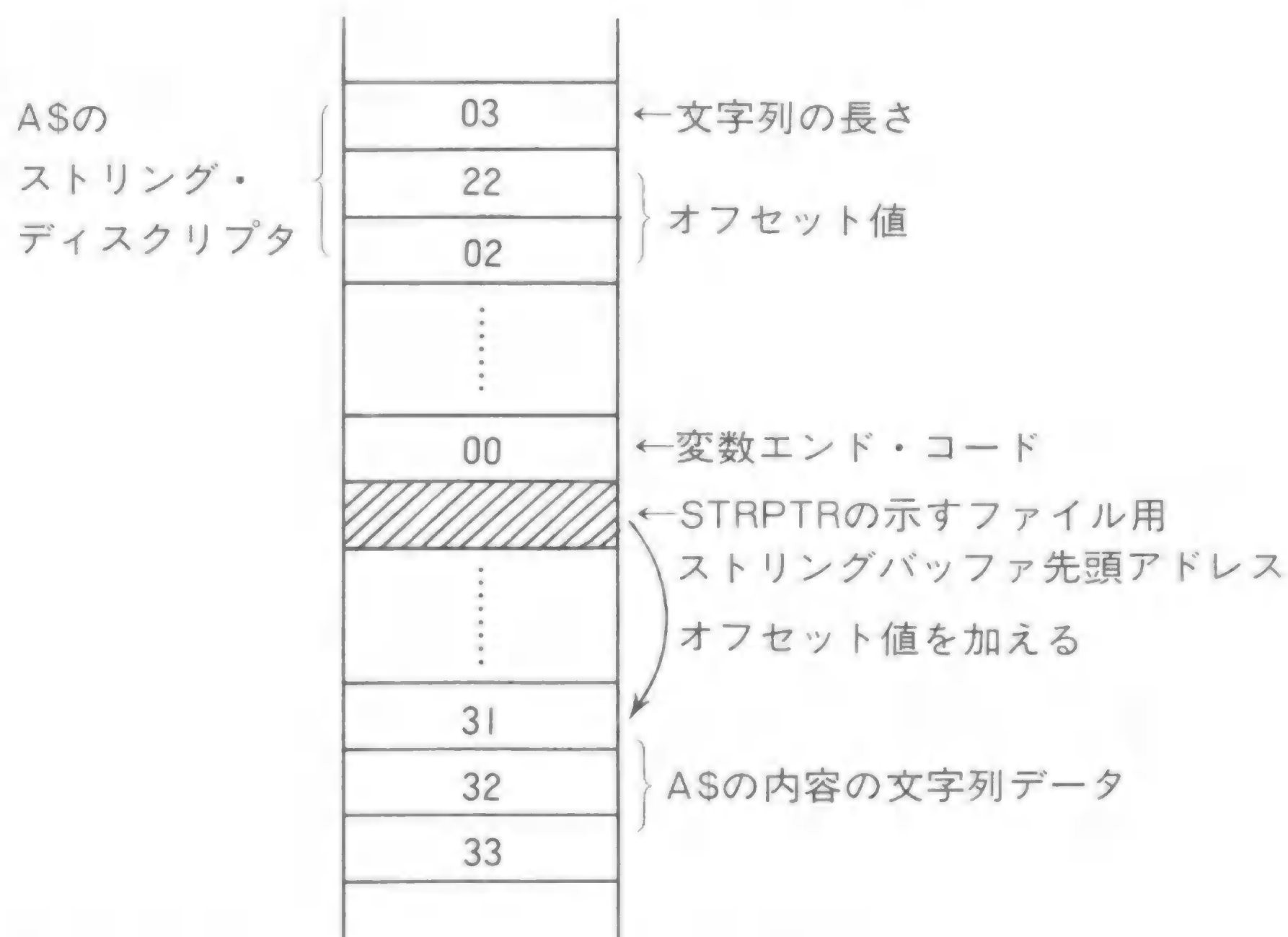
数値型と最大の違いは、変数名の後のデータ部です。数値変数では、この部分に実際の数値データが入るのですが、文字列変数では**ストリング・ディスクリプタ**（string descriptor）とよばれる3バイト情報が書かれます。その内訳は以下のようです。

《ストリング・ディスクリプタ》

文字列の長さ ／ 文字列の格納されるアドレスのオフセット値（STRPTRを0とする）

たとえば、9FEBH〜9FEDHの3バイトは最初の03Hが変数A$に格納される文字列"123"の長さを示し、次の22H,02Hが実際に文字列のあるアドレスのオフセット値0222Hを示します。ここで、オフセット値とは、変数エリア直後の「ファイル用ストリング・バッファ」の先頭（STRPTR）を0とする相対アドレスです。ですから、9FF5H+0222H=A217Hが実際に文字列"123"の格納されているアドレスとなります。

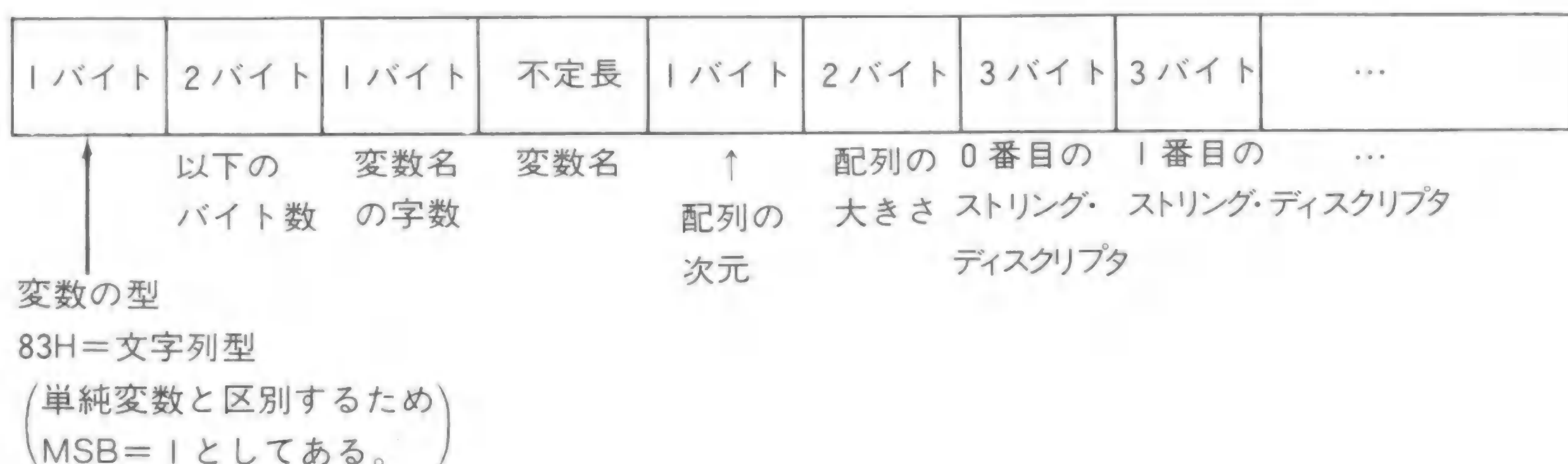
すなわち、文字列変数のデータ格納は、変数A$を例にとると次のようになります。

《文字列データの探し方》

	メモリ	
A$のストリング・ディスクリプタ	03	←文字列の長さ
	22	オフセット値
	02	
	⋮	
	00	←変数エンド・コード
	▨	←STRPTRの示すファイル用ストリングバッファ先頭アドレス
	⋮	オフセット値を加える
	31	A$の内容の文字列データ
	32	
	33	

以上は単純文字列変数の場合ですが、配列文字列変数（A$(1)など）の場合も察しがつくと思います。考え方は、配列数値変数の場合と同様です。

《配列文字列変数の格納形式》

1バイト	2バイト	1バイト	不定長	1バイト	2バイト	3バイト	3バイト	…
変数の型	以下のバイト数	変数名の字数	変数名	配列の次元	配列の大きさ	0番目のストリング・ディスクリプタ	1番目のストリング・ディスクリプタ	…

変数の型
83H＝文字列型
（単純変数と区別するため MSB＝1としてある。）

12−10 デバイス・テーブル

HuBASICでは、周辺装置(デバイス)を統一的に管理し、またバージョンアップ等にも速やかに対応できるように、**デバイス・テーブル**とよばれる表を内蔵しています。

各デバイスには、3文字以下のアルファベットと1文字の数字(0~9,**ユニット番号**という)からなる名前がつけられています。

《デバイス名》

名 称	ユニット番号	周　辺　装　置	モ　ー　ド
CRT		画面。ただしコントロール・コードが効かない場合である。	"O"のみ。(出力)
SCR		画面。ただしコントロール・コードが効く場合である。	"O"のみ。(出力)
KEY		キーボード。	"I"のみ。(入力)
LPT		プリンタ。	"O"のみ。(出力)
CAS	0～9	カセットテープ。ユニット番号は、意味をなさない(省略してもよい)。	"I"、"O"(入出力)
MEM		グラフィック・メモリー。OPTION SCREEN 2命令により、入出力デバイスとして使用できる。	"I"、"O"、"R"、"A"(ランダムアクセス、アペンドも可)
EMM	0～9	外部メモリー(I/Oデータ機器社のEMMボード)。EMM0:~EMM9:まで10個分用意されているが、ボードを装着しないと使用できない。(ユニット番号は省略可)	"I"、"O"、"R"、"A"
	0～3	フロッピー・ディスク。直接ユニット番号により0:～3:で指定する。ディスク装置は、4台つなぐことができる。テープBASICでは使用できない。	"I"、"O"、"R"、"A"

〔注〕モード欄は、入出力用途およびファイルモード指定を意味します。

- "O" (Output)……………………デバイスへの出力
- "I" (Input)……………………デバイスからの入力
- "R" (Random file)………ランダム・アクセスのファイル指定
- "A" (Append)…………………ファイルのアペンド(追加)指定

さて、これらのデバイスの1つ1つに対応して、HuBASICの内部に検索表が用意されています。次に掲げるのは、テープBASICにおけるデバイス"CAS:"のテーブルをダンプしたものです(ディスクBASICの方は、6A18H番地~6A47H番地をダンプして下さい)。

《デバイス“CAS：”のテーブル》

```
:5E17=96 62 43 41 53 00 32 61 /ｰbCAS.2a
:5E1F=79 60 73 5E 93 5E A0 5C /y`s^ｭ^ ¥
:5E27=B7 5E 9E 5E 42 60 2E 60 /ｷ^ﾞ^B`.`
:5E2F=3B 60 CF 5E 2A 60 A0 5C /;`ﾏ^*` ¥
:5E37=A0 5C A0 5C 60 5E 6D 5E / ¥ ¥`^m^
:5E3F=47 5E 05 61 4D 5E 66 5E /G^.aM^f^
```

デバイス・テーブルは各デバイスごとに48バイトからなり、すべて同一の形式をしています。表の読み方は以下の通りです。

デバイス・テーブルの内容

機能番号	テーブル内相対アドレス	意　　味
	00H～01H	次のデバイス・テーブルのアドレス(下位、上位の順)。
	02H～05H	デバイス名（３文字）＋00H
0番	06H～07H	FILES処理アドレス
1番	08H～09H	OPEN処理アドレス
2番	0AH～0BH	CLOSE処理アドレス
3番	0CH～0DH	KILL処理アドレス
4番	0EH～0FH	NAME処理アドレス
5番	10H～11H	1バイト入力処理アドレス
6番	12H～13H	1行入力処理アドレス
7番	14H～15H	1バイト出力処理アドレス
8番	16H～17H	TAB出力処理アドレス
9番	18H～19H	CR出力処理アドレス
10番	1AH～1BH	EOF（ファイル終了検出）処理アドレス
11番	1CH～1DH	POS（現在位置検出）処理アドレス
12番	1EH～1FH	DEVF（フリーエリア）処理アドレス
13番	20H～21H	LOF（ファイルサイズ）処理アドレス
14番	22H～23H	LOC（ファイル位置）処理アドレス
15番	24H～25H	LOAD処理アドレス
16番	26H～27H	SAVE処理アドレス
17番	28H～29H	VERIFY処理アドレス
18番	2AH～2BH	INIT（初期化。フォーマット）処理アドレス
19番	2CH～2DH	GET処理アドレス
20番	2EH～2FH	PUT処理アドレス

上記“CAS：”のテーブルの場合、最初の２バイトは次のテーブルが、6296番地から格納されていることを示します。次の４バイト　43 H,41 H,53 H,00 H はデバイス名 CAS の文字列です(00 H＝エンドマーク)。5 E 1 DH番地からは、処理アドレスが２バイトずつ上下位逆転して格納されています。たとえば、FILES“CAS:”処理は、6132 H番地～を見ればよいことがわかります。(ただし０番～20番の機能の中には、サポートされてないものもあり、その場合は、エラー処理アドレスが書かれています。)

12−11　ファイル用ストリング・バッファ

前節で、デバイス・テーブルの説明をしたのは、先ほど来、テキストエリアや変数エリアの後ろに謎めいて出現していた「ファイル用ストリング・バッファ」とよばれるエリアについて考えるためです。

このエリアは本来、カセットテープやフロッピーディスク等の補助記憶装置とファイル単位でのデータの読み書きをするためにメインメモリー上に設けられた領域（バッファ）です。各ファイルには、＃1、＃2 などの番号がつけられていますが、1つのファイルに対して、16バイトのインフォメーション部（ファイル・コントロール・ブロック）、256バイトのデータ部、計272バイト分がバッファとして設けられ、補助記憶装置との中継地点の役割を果たします。

同時に扱うことのできるファイルは、＃1～＃15 の最大15個で、上限は MAXFILES 命令で指定されます。 MAXFILES n を実行すると、＃0～＃n の計 (n+1) 個分のファイル・バッファが設けられます。メモリーマップに「(n+1)×(16+256)バイト」とあるのはその意味です。BASIC起動時には、ファイル番号の最大値は1に指定されますが、ディスク BASIC では、システム・ディスク内の "Start up" プログラムが実行されて、MAXFILES 2 に指定されます。

ファイル・バッファの ＃0 は、システムが入出力管理のために使用するエリアです。本章の最初の方「テキストの格納形式」で述べたように、ファイル・バッファ＃0の最初の数バイト分は、入出力装置の情報が書かれています。

```
00H,  4FH,  E5H,  5DH,  ………………
 ↑      ↑    └─────┘
ファイル属性  モードは   デバイスLPTのテーブル
(KILL=      "O"       アドレスで、LLISTを実
 無意味)               行したからである。
```

本章第2節で触れた上のようなコード列は、プログラムリストをプリンタ出力したことにより生じたものです。

ファイル・バッファの ＃1～ に関しては、私自身まだ不明な点を残していますので、いつの日か専門家の方が解析して下さるのを期待して、今は触れないことにしたいと思います。

12−12 各種のポインタ・アドレス

テキストエリア、変数エリア、ファイル用ストリング・バッファ等は、プログラムの実行や変更、入力等に伴い、ダイナミックに変化していきますが、各エリアの先頭アドレス、終了アドレス等は、インタプリタ内の特定のアドレスに格納されています。

エリア	ポインタ格納アドレス
テキスト・エリア	←9D52H、9D53Hに格納（ディスクBASICでは、A635H、A636H）
変数エリア	←9D46H、9D47Hに格納（A629H、A62AH）
ファイル用ストリング・バッファ	←9D48H、9D49Hに格納（A62BH、A62CH）
ストリング・データバッファ	←35F6H、35F7Hに格納（3628H、3629H）
テンポラリー（一時的）ストリング・エリア	←9D4AH、9D4BHに格納（A62DH、A62EH）
フリー・エリア	←35EFH、35F0Hに格納（3621H、3622H）

〔注〕（　　）はディスクBASICでのアドレス。

HuBASICのシステム変数である **STRPTR**（String pointerの略）は、9D48H、9D49H番地の内容を持つものです（ディスクBASICでは、A62BH、A62CH番地）。

たとえば、次のプログラムを入力して、各々のポインタ・アドレスを見てみましょう（ディスクBASICの方は、アドレスを変えて下さい）。

《ポインタを見る》

```
LIST
10 A=16
20 B$="CDE"
30 END
Ok
RUN
Ok
MON
*D 9D46 9D52
:9D46=E4 9F F3 9F 1A A2 00 FC /◆\木\.「.円
:9D4E=00 FD 00 FF C5 9F C4 00 /.人.テナ\ト.
*M 35EF
:35EF=1A
:35F0=A2
*M 35F6
:35F6=13
:35F7=A2
*■
```

この例では、メモリーマップは次のようになっていることがわかります。

9FC5H	テキスト・エリア
9FE4H	変数エリア
9FE3H	ファイル用 ストリング・バッファ
A213H	ストリング・データ バッファ
A21AH	テンポラリー ストリング・エリア
A21AH	フリーエリア

この情報に従って、実際にメモリーをダンプすると、次のようになっているはずです。今まで理解したことの総まとめとして御覧下さい。

```
:9FC5=0A 00 0A 00 41 F4 12 10 /....A木..
:9FCD=00 00 0D 00 14 00 42 24 /......B$
:9FD5=F4 22 43 44 45 22 00 06 /木"CDE"..
:9FDD=00 1E 00 98 00 00 00 05 /...┘....
:9FE5=01 41 85 00 00 00 00 03 /.A■.....
:9FED=01 42 03 22 02 00 00 41 /.B."...A
:9FF5=85 00 00 00 00 03 01 42 /■......B
:9FFD=03 22 02 00 00 4F 39 5D /."...O9]
:A005=00 00 00 00 00 00 00 00 /........
:A00D=00 00 00 00 00 00 00 00 /........
:A015=00 00 00 00 00 00 00 00 /........
:A01D=00 00 00 00 00 00 00 00 /........
:A025=00 00 00 00 00 00 00 00 /........
:A02D=00 00 00 00 00 00 00 00 /........
:A035=00 00 00 00 00 00 00 00 /........
:A03D=00 00 00 00 00 00 00 00 /........

:A213=0B 00 43 44 45 00 00 00 /..CDE...
```

HuBASICには、STRPTR の他に、ポインタ・アドレスを返す関数として、VARPTR（Variable pointer の略） があります。この関数は、 VARPTR（数値変数） とすると、その数値変数に入っているデータの格納アドレスを返します。上の例では、 ?HEX$(VARPTR(A)) とすると、変数Aの内容である単精度16（の内部表現）が格納されているアドレス9FE7H番地が返ってくるはずです。

注意すべきなのは、文字列変数の場合で、 VARPTR(文字列変数) では、その文字列変数のデータ内容ではなく、ストリング・ディスクリプタ（3バイト）の格納アドレスが返ってくる点です。上記の例では、 ?HEX$(VARPTR(B$)) とすると、変数B$のストリング・ディスクリプタの格納アドレスである9FEFH番地が返ってきます。この点、マニュアルの記載がわかりにくいので、皆さんでいろいろ実験してみて下さい。

12−13 USR関数の概要

さて、本章の最後は、BASICからマシン語プログラムを呼び出すためのユーザー関数USRについて考察したいと思います。

BASICから、マシン語サブルーチンを呼び出す命令としては、CALL命令とUSR関数が設けられていますが、USR関数の方は、BASICの変数とマシン語サブルーチンとの間でデータの授受ができるので、これに習熟すると、高度なプログラムを書くことができます。USR関数は後ろに番号をつけて、USR 0〜USR 9の計10個まで区別できます。番号を省略するとUSR 0と同じと見なされます。

USR関数は、使用に先立って、DEFUSR＝コール・アドレス あるいは DEFUSRn＝コール・アドレス の形で、呼び出すサブルーチンのアドレスを登録しておく必要があります。一度このように定義すると、インタプリタ内の「DEFUSRテーブル」にコール・アドレスが書き込まれます。

```
DEFUSR0=&HFD00
Ok
MON
*D 9D32 9D45
:9D32=00 FD 9F 9C 9F 9C 9F 9C
:9D3A=9F 9C 9F 9C 9F 9C 9F 9C
:9D42=9F 9C 9F 9C E4 9F C8 9F
*■
```

テープBASICでは、9 D 32 H〜9 D 45 H番地の2×10バイト分に、USR 0〜USR 9のコール・アドレスが登録されます(ディスクBASICでは、A 615 H〜A 628 H番地)。初期値として登録されている 9 C 9 FH (ディスクではA582H) というアドレスは、Undefined function のエラー表示をするルーチンのアドレスです。このテーブルは一度登録されると、次に再登録するまで消えませんから、マニュアルに「指定せずに使用する事はしないで下さい。定義を解除する命令はありませんので、前の定義がのこったままとなり、暴走する事があります。」と書かれている注意が大切になる訳ですね。

ではUSR関数の実験にかかります。まず、USRによりマシン語サブルーチンがコールされた時、各レジスタの内容がどうなっているかを見るために、レジスタ表示のプログラムを利用します。次にアセンブラ・ソース・リストを掲げますので、CLEAR &HFC 00により、マシン語フリーエリアを確保してから、マシン語部分をモニターより入力して下さい(プログラムはFC 00 H〜FC 40 H番地に格納されます)。

Registers Display Program(0)

```
0000            ;*************************************************
0001            ;*                                               *
0002            ;*        Registers Display Program (0)          *
0003            ;*                                               *
0004            ;*   ( used subroutines in Hu BASIC monitor )    *
0005            ;*                                               *
0006            ;*            by Yasuhiro Shimizu                *
0007            ;*               1984.1.6 FRI                    *
0008            ;*                                               *
0009            ;*************************************************
0010            ;
0011 0013       OUTCRT:EQU   0013H          ;print 1 character
0012 1202       PRHL:  EQU   1202H          ;print HL-reg
0013 04A7       CRLF:  EQU   04A7H          ;let new line
0014 04BA       PRSPC: EQU   04BAH          ;print space
0015            ;
```

```
0016                          ORG   0FC00H
0017                    ;
0018 FC00 FDE5          MAIN:  PUSH IY          ;*** registers store to stack ***
0019 FC02 DDE5                 PUSH IX
0020 FC04 E5                   PUSH HL
0021 FC05 D5                   PUSH DE
0022 FC06 C5                   PUSH BC
0023 FC07 F5                   PUSH AF
0024                    ;
0025 FC08 CDA704               CALL CRLF        ;let new line
0026                    ;
0027 FC0B 061B          PRNAME:LD   B,27        ;*** print register name 'AF-IY' *
0028 FC0D 2126FC               LD   HL,DATA     ;HL = data top address
0029 FC10 7E            LOOP1: LD   A,(HL)      ;27 characters 'AF-IY'
0030 FC11 CD1300               CALL OUTCRT
0031 FC14 23                   INC  HL
0032 FC15 10F9                 DJNZ LOOP1
0033                    ;
0034 FC17 CDA704               CALL CRLF
0035                    ;
0036 FC1A 0606                 LD   B,6         ;*** print 6 register contents ***
0037 FC1C E1            LOOP2: POP  HL          ;6 registers 'AF-IY'
0038 FC1D CD0212               CALL PRHL
0039 FC20 CDBA04               CALL PRSPC
0040 FC23 10F7                 DJNZ LOOP2
0041                    ;
0042 FC25 C9                   RET              ;return
0043                    ;
0044 FC26 41462020 DATA:  DB   'AF   BC   DE   HL   '
     FC2A 20424320
     FC2E 20204445
     FC32 20202048
     FC36 4C202020
0045 FC3A 49582020             DB   'IX   IY'
     FC3E 204959
0046                    ;
0047 FC41                      END
```

準備ができましたら、 DEFUSR＝&HFC 00 により USR 関数を定義して、実験開始です。

USR 関数は、あくまで「関数」ですから、その起動には「引数」を必要とします。また、USR 関数は値を返してきますから、それを何かの変数で受けとっておきましょう。ここでは、USR の頭文字の変数Uを用いることにします。かくして、USR 関数によるマシン語サブルーチンの呼び出しは、

U＝USR(数値)　あるいは
U＄＝USR(文字列)

※U, U$である必要はなく他の変数を用いてもかまいません。

という代入文の形をとることになります。 USR(文字列) では、ふつう USR 関数は文字列を返してきますから、この場合は「文字列変数」U＄で受けることにします。様々なデータを「引数」にして実験してみて下さい。以下に、数値の例と文字列の例を挙げておきます（テープ BASIC での実験データです)。

《USRコール時のレジスタ》

```
DEFUSR=&HFC00
Ok
U=USR(65536!)
AF   BC   DE   HL   IX   IY
0502 0008 FB08 FB00 2064 0000
Ok
U$=USR("ABC")
AF   BC   DE   HL   IX   IY
0370 0308 A1F8 FB00 2064 0000
Ok
```

実験からわかることは以下の事柄です。

データ型	Aレジスタ	HLレジスタ	IXレジスタ	Bレジスタ	DEレジスタ
整　　数	02H	FB00H	2064H	00H	FB08H
単精度実数	05H	FB00H	2064H	00H	FB08H
倍精度実数	08H	FB00H	2064H	00H	FB08H
文　字　列	03H	FB00H	2064H	文字列の長さ	文字列の格納アドレス

〔注〕ディスクBASICの方は、IXレジスタの内容が2076Hとなっているはずです。

この実験により、マニュアルの「USR命令の使い方」に書かれていることが検証されます。すなわち、USRによりマシン語サブルーチンが呼び出されると、次のデータがレジスタを経由して、サブルーチンに受け渡されることになるのです。

《USRによりサブルーチンへ受け渡されるデータ》

Aレジスタ	データの型を示す。 整数型………………02H 単精度実数型………05H 倍精度実数型………08H 文字列型……………03H
HLレジスタ	数値型データのとき データの内部表現が格納されているアドレスを示す。 文字列型データのとき データのストリング・ディスクリプタ（3バイト）の格納されているアドレスを示す。
IXレジスタ	エラー処理ルーチンのアドレスを示す。
Bレジスタ	数値型データのとき、つねに00Hとなる。 文字列型データのとき そのデータの長さを示す。
DEレジスタ	数値型データのとき HLレジスタの内容＋8を示す。 文字列型データのとき 文字列データの格納アドレスを示す。

次節以降、データの活用法を詳しく見ることにします。

12−14　浮動小数点アキュムレータ(FAC)

前節の実験で、データ格納アドレス（HLレジスタの内容）として FB 00 H が登場しましたが、本節のテーマはこのアドレスの解明です。

まず注意することは、HLレジスタの内容は必ずしも FB 00 H ではないということです。CLEAR&HF 000 として、前節の実験をすると、HLレジスタの内容は EF 00 H を示すはずです。

```
CLEAR &HF000
Ok
DEFUSR=&HFC00
Ok
U=USR(.1#)
AF   BC   DE   HL   IX   IY
0802 0808 EF08 EF00 2064 A8E8
Ok
U$=USR("123")
AF   BC   DE   HL   IX   IY
0370 0308 A1F8 EF00 2064 A8E8
Ok
```

HuBASICのコマンド CLEAR アドレス は、ユーザー用のマシン語フリーエリアの確保のために使われる命令です。「アドレス」は、BASICの使用するアドレスの上限＋１すなわち、フリーエリアの先頭アドレスを指定します。

さて、HLレジスタの話に戻りますが、USR の実験からわかることは、USR によりマシン語サブルーチンに渡されるHLレジスタの内容はつねに、（マシン語フリーエリア先頭アドレス−100 H）の値を示しているということです。本章冒頭のメモリーマップでも書いたように、このアドレスからは FAC というエリアになっています。このエリアは何でしょうか？　FAC とは、

Floating point ACcumulator あるいは
Floating ACcumulator

の略称で、**「浮動小数点アキュムレータ」**あるいは**「浮動アキュムレータ」**とよばれる領域です。

「数値の内部表現」の節でも述べたように、コンピュータでは、実数は浮動小数点形式で扱われることが多く、そしてHuBASICでは、これを２進データとして内部表現している訳ですね。実数の四則演算（加減乗除）は、これら浮動小数点形式のデータを対象とすることになりますが、Ｘ１のCPUである Z 80 A の内部にはこれを計算するためのレジスタ（アキュムレータ＝累算器）がありません。よって、浮動小数点演算はソフトウェア（プログラム）によって行なう必要があり、そのためのアキュムレータはワークエリアとしてメモリー上に確保しなければなりません。こうして設けられるエリアが「浮動小数点アキュムレータ」(FAC）である訳です。HuBASICでは、BASIC使用エリアの最後の256バイトを FAC にあてています。

ＵＳＲ関数により、マシン語サブルーチンが呼び出された時点で、HLレジスタは ＦＡＣ の先頭アドレスを指示しています。そして、サブルーチンに渡されるデータは FAC に転送されてきます。

ここで、Aレジスタに設定される内容を思い出して下さい。USRにより、Aレジスタには「データの型」を示すコードが入るのでした。このコードは、デタラメに決められているのではありません。FACに格納されるデータのバイト数を示しているのです。02H,05H,08Hは各々数値の内部表現に要するバイト数ですね。また、文字列型の03Hは、ストリング・ディスクリプタのバイト数3を示しています。

「数値の内部表現」の項で掲載したプログラムがありますが、あれは、FACの内容を先頭から8バイト分ダンプするプログラムだったのです。FACは、インタプリタ内部でも常に使用されていますから(行番号計算、座標アドレス変換、数値計算など)、FACの内容を見るには、プログラム中で直接画面表示するか、あるいは別の場所に転送しておかないと、BASICに戻った時インタプリタがFACを書きかえてしまうので注意しましょう。

12—15 USR関数の活用(1)

——整数データを渡す——

本節から3節にわたり、USR関数の活用例を見ていきます。まず最初は、USR関数によりマシン語サブルーチンに整数データを渡す方法です。ゲームでは、ほとんどの数値データは整数型ですから、USR関数による整数データの渡し方を知っておく必要がありますね。

これから掲げるサンプルプログラムは、画面の中程に11×5個のキャラクタを表示するものです。これくらいの数になると、BASICのPRINT文では、遅さのために先頭のキャラクタと末尾のキャラクタで時間のズレが生じ、歪むことがあります。こういう場合は表示部分をマシン語サブルーチン化すると、時間のズレはほとんど気にならなくなるので効果的です。

310行にUSR関数があります。引数VRAM%は整数型ですから、先頭キャラクタの表示位置のVRAMアドレスを2バイト整数データとして、マシン語サブルーチンへ渡します。サブルーチン側では、490行〜510行にあるように、FACの先頭から2バイト分をBCレジスタに受け取っています。

なおこのプログラムでは、テンキーの4と6によりキャラクタを左右に移動できるようにしていますから(320行〜340行)、動かしてみて下さい。プログラムは無限ループになっているので、止めるには SHIFT + BREAK を用いて下さい。

USR sample(1)—integer—

```
100 REM ****************************
110 REM *                          *
120 REM *   USR sample (1)         *
130 REM *                          *
140 REM *   -- integer --          *
150 REM *                          *
160 REM ****************************
170 '
180 CLEAR &HF000
190 ADR=&HF000 : RESTORE 490 : GOSUB "ｶｷｺﾐ" :REM  ｻﾌﾞﾙｰﾁﾝ
200 ADR=&HF100 : RESTORE 780 : GOSUB "ｶｷｺﾐ" :REM  ﾃﾞｰﾀ
210 DEFUSR=&HF000
220 '
230 REM ｼﾞｭﾝﾋﾞ ----------------
240 '
250 WIDTH 40 : INIT : CLS : CLICK OFF
260 X=20 : Y=10
270 '
280 REM ﾒｲﾝ --------------------
290 '
300 VRAM%=&H3000+Y*40+X
310 U=USR(VRAM%)  :REM print
320 I=STICK(0)
330 IF I=4 AND X>0  THEN X=X-1
340 IF I=6 AND X<29 THEN X=X+1
350 GOTO 300
360 '
370 LABEL "ｶｷｺﾐ" :REM----------
380 '
390 READ MC$
400 IF MC$="END" THEN RETURN
410 POKE ADR,VAL("&H"+MC$)
420 ADR=ADR+1
430 GOTO 390
440 '
450 REM ﾏｼﾝｺﾞ ﾃﾞｰﾀ ------------
460 '
470                     :REM            ORG   0F000H
480                     :REM   ;
490 DATA 4E             :REM   START:   LD    C,(HL)       ;BC = FAC data
500 DATA 23             :REM            INC   HL
510 DATA 46             :REM            LD    B,(HL)
520 DATA C5             :REM            PUSH  BC
530 DATA E1             :REM            POP   HL           ;HL = start VRAM adrs
540 DATA DD,21,00,F1    :REM            LD    IX,DATA
550 DATA 16,05          :REM            LD    D,5          ;line count = 5
560 DATA E5             :REM   LOOP1:   PUSH  HL
570 DATA C1             :REM            POP   BC           ;BC = initial VRAM adrs
580 DATA 1E,0B          :REM            LD    E,11         ;char count = 11
590 DATA DD,7E,00       :REM   LOOP2:   LD    A,(IX+0)     ;A  = data
600 DATA ED,79          :REM            OUT   (C),A        ;print
610 DATA CB,A0          :REM            RES   4,B          ;BC = attribute adrs
620 DATA 3E,06          :REM            LD    A,06H        ;yellow
630 DATA ED,79          :REM            OUT   (C),A
640 DATA CB,E0          :REM            SET   4,B          ;BC = VRAM adrs
650 DATA DD,23          :REM            INC   IX           ;data adrs inc
660 DATA 03             :REM            INC   BC           ;VRAM adrs inc
670 DATA 1D             :REM            DEC   E            ;counter dec
680 DATA 20,ED          :REM            JR    NZ,LOOP2
690 DATA 01,28,00       :REM            LD    BC,40
700 DATA 09             :REM            ADD   HL,BC        ;HL = HL+40
710 DATA 15             :REM            DEC   D            ;line count dec
720 DATA 20,E2          :REM            JR    NZ,LOOP1
730 DATA C9             :REM   EXIT:    RET
740 DATA END,           :REM   end mark (1)
750                     :REM   ;
760                     :REM            ORG   F100H
770                     :REM   ;
780 DATA 20,20,20,9A    :REM   DATA:    DB    '   ┌─┬─┐   '
790 DATA 90,93,90,97    :REM
800 DATA 20,20,20       :REM
810 DATA 20,9A,90,94    :REM            DB    ' ┌─┤♥│♥├─┐ '
820 DATA E3,91,E3,95    :REM
830 DATA 90,97,20       :REM
840 DATA 20,98,20,99    :REM            DB    ' ┘ └┬┴┬┘ └ '
850 DATA 93,92,93,98    :REM
860 DATA 20,99,20       :REM
870 DATA 20,20,20,20    :REM            DB    '    │ │    '
880 DATA 91,20,91,20    :REM
890 DATA 20,20,20       :REM
900 DATA 20,20,99,90    :REM            DB    '  └─┴ ┴─┘  '
910 DATA 92,20,92,90    :REM
920 DATA 98,20,20       :REM
930 DATA END,           :REM   end mark (2)
940 '
950 REM---------------------------------------------------------------
```

12—16 USR関数の活用（2）

——実数データを渡す——

本節の内容は、ゲームではほとんど使うことがないでしょうが、フライト・シミュレータのようなグラフィック使用時や、科学技術計算を高速化するには、知っておく必要があるでしょう。

まず、基本から行きましょう。USR関数により、三角関数 SIN の計算をしてみます。ここでは、BASICインタプリタ内蔵のシステム・サブルーチンを利用します。

名　　称	SINエントリー
アドレス	8BCCH番地（ディスクBASICでは94AFH番地）
入力条件	Aレジスタ　＝データの型（数値に限る） HLレジスタ＝FAC先頭アドレス FAC　　　＝データ（ラジアン値の内部表現）
出力条件	Aレジスタ　＝データ1バイト目（指数部） HLレジスタ＝FAC先頭アドレス FAC　　　＝SIN値の内部表現
機　　能	ラジアンで与えられた角度のSINを計算する。

《USR関数によるSIN計算》

```
100 REM **************************
110 REM *                        *
120 REM *   USR sample (2-1)     *
130 REM *                        *
140 REM *   sin calculation      *
150 REM *                        *
160 REM **************************
170 '
180 DEFUSR=&H8BCC  :REM  sin entry
190 '
200 INPUT "angle(°) = ";A!
210 Y=USR(RAD(A!))
220 PRINT Y
230 GOTO 200
```

```
RUN
angle(°) = ? 30
 .5
angle(°) = ? 120
 .8660254
angle(°) = ? -90
-1
angle(°) = ? 0
 0
angle(°) = ? ■
```

ユーザー自身が、浮動小数点演算を行なうルーチンを自作するのは、かなり大変なことです。HuBASIC内部には、浮動小数点演算のルーチンが基本パッケージとして用意されてますから、これらを利用しない手はありません。(元気のある読者は、これらのルーチンを解析してみて下さい。かなり長く複雑なプログラムです。)

さて、今度は入力した2つの単精度数値をかけ算して、出力するプログラムをUSR関数を利用して作ってみましょう。使用するシステム・サブルーチンは次のものです。

名　　　称	浮動小数点乗算ルーチン
ア　ド　レ　ス	9712H番地（ディスクBASICでは、9FF5H番地）
入　力　条　件	Aレジスタ　＝データの型（数値に限る） HLレジスタ＝第1データ先頭アドレス DEレジスタ＝第2データ先頭アドレス (HL)～　　＝第1データの内部表現 (DE)～　　＝第2データの内部表現
出　力　条　件	HLレジスタ＝結果の先頭アドレス (HL)～　　＝(第1データ)×(第2データ)の内部表現
機　　　能	2つの数値データの乗算を行なう。

以下のサンプルプログラムはテープBASICを想定してあります。ディスクBASICの方は、マシン語サブルーチン中500行を DATA　CD,F5,9F に変更すると全く同じに動くはずです。250行～270行の処理は、おわかりですか？ 入力数値データの一方の格納アドレスを、サブルーチンで参照するために、ワークエリア F100H,F101H に保護しておきます。サブルーチンでは、490行でこのアドレスをDEレジスタに戻します。VARPTR関数の使い方をよく覚えて下さい。

```
100 REM ****************************
110 REM *                          *
120 REM *    USR sample (2-2)      *
130 REM *                          *
140 REM *     multiplication       *
150 REM *                          *
160 REM ****************************
170 '
180 CLEAR &HF000
190 GOSUB "ｶｷｺﾐ"
200 DEFUSR=&HF000
210 '
220 LABEL "ﾆｭｳﾘｮｸ" :REM ------------------------------------------
230 '
240 INPUT "(DE)=";DE!
250 ADR$=HEX$(VARPTR(DE!))
260 POKE &HF100,VAL("&H"+RIGHT$(ADR$,2)) :REM store VARPTR(DE!)
270 POKE &HF101,VAL("&H"+LEFT$(ADR$,2))
280 '
290 INPUT "(HL)=";HL!
300 '
310 REM -- ｶｹｻﾞﾝ ---------------------------------------------
320 '
330 U=USR(HL!)
340 PRINT "(HL)=(HL)*(DE) —> (HL)=";U
350 PRINT
360 GOTO "ﾆｭｳﾘｮｸ"
370 '
380 LABEL "ｶｷｺﾐ" :REM ---------------------------------------------
390 '
400 ADR=&HF000
410 READ MC$
```

```
420 IF MC$="END" THEN RETURN
430 POKE ADR,VAL("&H"+MC$)
440 ADR=ADR+1
450 GOTO 410
460 '
470 REM -- ﾏｼﾝｺﾞ ｻﾌﾞﾙｰﾁﾝ -----------------------------------------------
480 '
490 DATA ED,5B,00,F1   :REM  LD   DE,0F100H  ;DE=VARPTR(DE!)
500 DATA CD,12,97      :REM  CALL 9712H      ;multiple real numbers
510 DATA C9            :REM  RET
520 DATA END,          :REM  end mark
530 '
540 REM -----------------------------------------------------------
```

```
RUN
(DE)=? 2
(HL)=? 5
(HL)=(HL)*(DE)  ―→ (HL)= 10

(DE)=? -5
(HL)=? .6
(HL)=(HL)*(DE)  ―→ (HL)=-3

(DE)=? .1
(HL)=? .1
(HL)=(HL)*(DE)  ―→ (HL)= .01

(DE)=? ■
```

HuBASIC内にはこの他にも、浮動小数点除算、加算,減算などの基本ルーチンが用意されています(付録のエントリーアドレス一覧表を参照)。これらを組み合わせれば、ユーザー独自の計算ルーチンを作ることもできるでしょう。

12−17 USR関数の活用法(3)

――文字列データを渡す――

USR関数活用篇の最後は、文字列データをマシン語サブルーチンへ渡す方法です。サンプルプログラムとして、入力文字列の順序を反転表示するサブルーチンを採り上げます。

USR(文字列変数) としてサブルーチンをコールすると、Bレジスタに文字列データの長さ、DEレジスタに文字列データの格納アドレスが設定されるのですが、サブルーチン内では、DEレジスタとHLレジスタを交換して、ポインタとしてより便利なHLをデータ・ポインタにしています(540行)。文字列反転のからくりは、550行〜580行および610行を見て下さい。

USR sample(3)――string――

```
100 REM *************************
110 REM *                       *
120 REM *    USR sample (3)     *
130 REM *                       *
140 REM *    ― string ―         *
150 REM *                       *
160 REM *************************
170 '
180 CLEAR &HF000
190 GOSUB "ｶｷｺﾐ"
200 DEFUSR=&HF000
210 '
220 REM -- ﾀｲﾄﾙ ----------------------------------------------------------
230 '
240 CLS
```

```
250 PRINT "────────────"
260 PRINT "ﾓｼﾞﾚﾂ ﾊﾝﾃﾝ"
270 PRINT "────────────"
280 PRINT
290 '
300 REM -- ﾒｲﾝ ------------------------------------------------------------
310 '
320 INPUT "ﾓｼﾞﾚﾂ = ";X$
330 PRINT "ﾊﾝﾃﾝ  =   "; : U$=USR(X$)
340 PRINT
350 GOTO 320
360 '
370 '
380 LABEL "ｶｷｺﾐ" :REM ------------------------------------------------------
390 '
400 ADR=&HF000
410 READ MC$
420 IF MC$="END" THEN RETURN
430 POKE ADR,VAL("&H"+MC$)
440 ADR=ADR+1
450 GOTO 410
460 '
470 REM -- ﾏｼﾝｺﾞ ｻﾌﾞﾙｰﾁﾝ ---------------------------------------------------
480 '
490                     :REM  ; string reverse`( B=length, DE=data adrs )
500                     :REM  ;
510                     :REM  OUTCRT: EQU  0013H       ;print 1 character
520                     :REM  CRLF:   EQU  04A7H       ;let new line
530                     :REM  ;
540 DATA EB             :REM  STRREV: EX   DE,HL       ;HL = top adrs of data
550 DATA 58             :REM          LD   E,B
560 DATA 16,00          :REM          LD   D,0         ;DE = length of string
570 DATA 1B             :REM          DEC  DE
580 DATA 19             :REM          ADD  HL,DE       ;HL = end adrs of data
590 DATA 7E             :REM  LOOP:   LD   A,(HL)      ;A  = character
600 DATA CD,13,00       :REM          CALL OUTCRT      ;print
610 DATA 2B             :REM          DEC  HL          ;data adrs increment
620 DATA 10,F9          :REM          DJNZ LOOP
630 DATA CD,A7,04       :REM          CALL CRLF        ;let new line
640 DATA C9             :REM  EXIT:   RET
650 DATA END,           :REM  end mark
660 '
670 REM ------------------------------------------------------------------
```

```
────────────
ﾓｼﾞﾚﾂ ﾊﾝﾃﾝ
────────────

ﾓｼﾞﾚﾂ  = ? 123456
ﾊﾝﾃﾝ   =   654321

ﾓｼﾞﾚﾂ  = ? ABCDE
ﾊﾝﾃﾝ   =   EDCBA

ﾓｼﾞﾚﾂ  = ? ■
```

以上、USR関数によるマシン語サブルーチンへのデータ引き渡しに関して、かなり詳細に見てきました。基本テクニックはこれらの中にすべて入っていますから、読者の皆さんでさらに面白い使い方を考えて下さい。

皆さん、USR関数なんてコワクありません！ 大いにUSRに親しもうではありませんか。私たち user の名を冠した関数なのですから。

付　録

付録1 SHARP HuBASIC エントリー・アドレス一覧

● SHARP　HuBASICは

テープBASIC　CZ-8CB01 Ver 1.0

ディスクBASIC　CZ-8FB01 Ver 1.0

の2つを対象としています。

● テープBASICとディスクBASICとで、エントリー・アドレスの配列順が一部前後しますが、順序は、ディスクBASICに準拠しています。

● 本資料作成にあたり、雑誌「マイコン」1983年2月号および1984年3月号に掲載の高橋雄一氏の記事を参考にさせていただきました。

IOCSブロック

テープBASIC	ディスクBASIC	機能
0000	0000	システム・スタート (JP 00FAH)
0003	0003	1行入力 (JP 017CH)
000B	000B	メッセージ表示 (JP 0483H)
0013	0013	1文字表示 (JP 04BCH)
001B	001B	1文字入力 (JP 029DH)
0023	0023	サブCPUとの入出力 (JP 0E07H)
002B	002B	PCGデータ・セット (JP 0AAAH)
0033	0033	CGデータ・リード (JP 0ACAH)
003B	003B	カセット・インフォメーション・ブロックSAVE (JP 0B75H)
003E	003E	カセット・データ・ブロックSAVE (JP 0B79H)
0041	0041	カセット・インフォメーション・ブロックLOAD (JP 0B9AH)
0044	0044	カセット・データ・ブロックLOAD (JP 0B9EH)
0047	0047	カセット・データ・ブロックVERIFY (JP 0BAEH)
004A	004A	BREAKチェック (JP 0330H)
004D	004D	WIDTH設定 (JP 0988H)
0066	0066	マスク不能割り込み (JP 00FAH)
00FA	00FA	システム初期化ルーチン
012D	012D	割り込みベクトル設定処理
013C	013C	PSG初期化。音声ミュート。
015A	015A	BASIC時、INPUT文用1行入力 (プロンプトは入力せず)
017C	017C	1行入力
029D	029D	1文字入力
02A6	02A6	INKEY$(2)処理
02AA	02AA	INKEY$(1)処理 (カーソル点滅1文字入力)
030C	030C	INKEY$処理
031D	031D	INKEY$(0)処理 (リアルタイム・キースキャン)
0330	0330	カセット走行チェック
0346	0346	キー入力割り込み処理 ※03C4Hでクリック音発生
03D3	03D3	割り込み処理からのリターン (RETI)
0483	0483	メッセージ表示 (DE=ポインタ、00H=エンド)

テープBASIC	ディスクBASiC	機　　種
04A7	04A7	1行改行
04AB	04AB	画面TAB実行
04BA	04BA	スペース1文字表示
04BC	04BC	1文字表示（コントロール・コード実行）
04C8	04C8	1文字表示（フォント出力）
0521	0521	行連続フラグ・テーブルのサーチ
0529	0529	（1行入力時の）行挿入処理（論理的2行目にかかるときも）
054A	054A	カーソル位置のVRAM絶対アドレス計算
0556	0556	カーソル行左端の相対アドレス計算
0577	0577	コントロールコード実行
05DE	05DE	CTRL＋N処理（カーソルから上をスクロール）
0610	0610	CTRL＋O処理（カーソルから下をスクロール）
069B	069B	（1画面）スクロール・アップ
06BF	06BF	CTRL＋K処理（HOME）
06E4	06E4	CTRL＋L処理（画面クリア）
071B	071B	CTRL＋J処理（行の分離）
071E	071E	CTRL＋C、M処理
073A	073A	↓処理
0740	0740	→処理
075F	075F	↑処理
0764	0764	←処理
0781	0781	CTRL＋W処理（次行との結合）
078B	078B	CTRL＋B処理（カーソルを1ワード左へ）
07A8	07A8	CTRL＋F処理（カーソルを1ワード右へ）
07F1	07F1	CTRL＋E処理（カーソルから右を1行消去）
07F7	07F7	CTRL＋G処理（ベル）
0814	0814	CTRL＋H処理（DEL）
08A1	08A1	CTRL＋I処理（HTAB）
08E4	08E4	CTRL＋R処理（INS）
0934	0934	CTRL＋T処理（水平TAB設定）
093D	093D	CTRL＋Y処理（水平TABとり消し）
094A	094A	CTRL＋Z処理（カーソル以下クリア）
098C	098C	WIDTH40処理
0998	0998	WIDTH80処理
09C0	09C0	出力ページ切り替え
09F5	09F5	入力ページ切り替え
0A3F	0A3F	画面初期化（CTRL＋D）
0A5A	0A5A	パレット、プライオリティ設定サブルーチン
0A6B	0A6B	テキスト画面クリア
0A8A	0A8A	OPTION SCREENチェック
0A8F	0A8F	グラフィック3画面クリア（CLS 0 相当）
0AAA	0AAA	PCGデータ・セット
0ACA	0ACA	CGデータ・リード
0B49	0B49	サブCPUからの1バイト入力
0B54	0B54	サブCPUへの1バイト出力
0B61	0B61	サブCPUからの入力準備待ち
0B6B	0B6B	サブCPUへの出力準備待ち
0B75	0B75	カセット・インフォメーション・ブロックSAVE
0B79	0B79	カセット・データ・ブロックSAVE
0B9A	0B9A	カセット・インフォメーション・ブロックLOAD
0B9E	0B9E	カセット・データ・ブロックLOAD
0BAE	0BAE	カセット・データ・ブロックVERIFY
0BBD	0BBD	テープへの書き込み処理
0C8A	0C8A	テープへの1バイト出力
0CAA	0CAA	テープからの1バイト入力
0CD6	0CD6	チェック・サム・バッファのクリア
0CDF	0CDF	テープへのGAP書き込み
0D15	0D15	テープ走行チェック

テープ BASIC	ディスク BASIC	機能
0D20	0D20	テープからのGAP読みとり
0D5C	0D5C	テープのREAD DATA立ち上がり検出
0D6D	0D6D	テープ空読み（約8秒間）
0D8A	0D8A	テープへの短パルス書き込み
0DA4	0DA4	テープへの長パルス書き込み（チェックサム付）
0DA5	0DA5	〃（チェックサムなし）
0DBF	0DBF	時間待ち（約166.25us）
0DC7	0DC7	カセット・モーター ON
0DEA	0DEA	カセット・モーター OFF
0DEC	0DEC	カセット制御ルーチン
0DF6	0DF6	テープ状態の読みとり
0DFE	0DFE	サブCPUへの2バイト出力
0E07	0E07	サブCPUへの入出力（汎用）
0E64	0E64	PSGからの4バイト入力（HL＝ポインタ）
0E77	0E77	PSGへの4バイト出力（HL＝ポインタ）
0E8A	0E8A	時間待ち（BC＝カウンタ）

モニター・ブロック

テープ BASIC	ディスク BASIC	機能
0FE2	0FE2	MONエントリー
1000	1000	モニター・ホット・スタート（SP＝ 0000H）
1003	1003	〃（SP初期化せず）
102D	102D	モニター時エラー処理
1061	1061	＊Pコマンド処理（プリンタ・スイッチON／OFF）
106A	106A	＊Sコマンド処理（カセットへのSAVE）
1095	1095	カセット停止
109A	109A	＊Lコマンド処理（カセットからのLOAD）
10B5	10B5	モニター用LOADサブルーチン
10E1	10E1	＊Vコマンド処理（カセットのVERIFY）
10F0	10F0	＊Rコマンド処理（BASICへのリターン）
10FD	10FD	モニター入力、スクリーン・エディット処理
111F	111F	16進キャラクタ・コード4文字→2バイト16進数値変換
1143	1143	16進キャラクタ・コード1文字→16進数値変換（00～0FH）
115E	115E	16進キャラクタ・コード2文字→1バイト16進数値変換
1186	1186	＊Gコマンド処理（ゴーサブ）
118B	118B	＊Dコマンド処理（メモリー・ダンプ）
11AF	11AF	メモリー・ダンプ用サブルーチン
1202	1202	HLの内容を16進4桁表示
1207	1207	Aレジスタの内容を16進2桁表示
121D	121D	＊Mコマンド処理（メモリー・セット）
124A	124A	"＝"表示
124E	124E	"："表示
1253	1253	＊Fコマンド処理（ファインド）
1292	1292	データ列比較（DE,HL＝ポインタ）
12A6	12A6	L,S,T用ファイル・アドレスをレジスタにセット（HL＝先頭アドレス、DE＝実行アドレス、BC＝バイト数）
12BE	12BE	＊Tコマンド処理（トランスファ）
12D5	12D5	プリンタへの改行出力
12DC	12DC	プリンタへの1文字出力（ヘッド位置ワークを＋1する）
12E2	12E2	プリンタへの1文字出力（ヘッド位置ワークは不変）
1315	1315	プリンタへのTAB出力
1321	1321	ファイルネームを画面表示
1340	1340	文字列を画面表示（HL＝ポインタ、D＝バイト数）
1349	1349	"(コーテーション）表示
134E	134E	ファイル・ネームの一致確認（＊L,＊V用）
138B	138B	ファイル・ネームをモニター時ファイル・バッファに転送

テープ BASIC	ディスク BASIC	機　　　能
13E9	13E9	パスワード作成（無指定なら20H）
1401	1401	パスワードをモニター時ファイル・バッファに書く。
1405	1405	タイマー・データをモニター時ファイル・バッファに書く。
141B	141B	ファイル属性不一致エラー表示 （モニター時、ERR? BASIC時、Bad file descripterエラー）
1420	1420	画面またはプリンタへの1文字表示（＊P指定）
142F	142F	画面またはプリンタへのメッセージ表示（＊P指定）
1446	1446	画面またはプリンタへの改行出力（＊P指定）
1450	1450	英小文字→英大文字変換
1478	1478	モニター時エラー処理（カセット停止つき）

インタプリタ・ブロック(1)

テープ BASIC	ディスク BASIC	機　　　能
14A0	14A0	BASICコールド・スタート（メモリー・クリアをする）
14C2	14CC	BASICホット・スタート（リセット時）
14DA	14EC	BASICホット・スタート（通常）
1507	1519	BASICプログラム入力ループ（OK表示なし）
151C	152E	CTRL＋Dによる初期化（BASICワーク・エリア、PSG）
1522	1534	BREAK時処理
153F	1551	CTRL＋Dによる周辺初期化（パレット、コンソールなど）
156C	157E	通常入力処理
1594	15A6	プログラムがもうない時の処理
15AD	15BF	LOAD、Rによるオートスタート
15B7	15C9	REM、ELSE、'エントリー 1行終了処理
15D3	15E5	1行実行およびダイレクト・モード実行
1657	166A	LETエントリー、代入文処理
1676	1689	ストリング代入処理
167E	1691	倍精度実数代入処理
1689	169C	単精度実数代入処理
168E	16A1	整数代入処理
1705	1713	ストリング・バッファ内文字列の消去
1758	1773	次の行の始まるアドレスを得る
1766	1781	ステートメント実行
178D	17A8	ステートメントのエンド・チェック（エンド・マークは00Hか3AH）
1795	17B0	RUNエントリー
17A9	17C4	RUN時の初期化
17D6	17F1	ON KEY GOSUBの指定解除
17ED	1808	FORエントリー
19AD	19C8	NEXTエントリー
1A66	1A81	PRINT、WRITE共通出力デバイスのセレクト
1AA5	1AC0	WRITEエントリー
1B04	1B1F	LPRINTエントリー
1B0C	1B27	PRINTエントリー
1B8F	1BAA	PAUSEエントリー
1BEF	1C0A	PRINT USING処理
1F80	1F9B	Format overエラー
1F85	1FA0	区切り記号のチェック
1F99	1FB4	STOPエントリー
1FA2	1FBD	LOAD時 BREAK処理
1FAF	1FCA	一般的BREAK処理
1FF3	2005	CONTエントリー
200D	201F	Out of memoryエラー（カセット停止、CLRつき）
2015	2027	Out of memoryエラー（表示のみ）

テープ BASIC	ディスク BASIC	機　　能
2018	202A	NEXT without FORエラー
201B	202D	Undefined labelエラー
201E	2030	Duplicate Definitionエラー
2021	2033	Subscript out of rangeエラー
2024	2036	RESUME without errorエラー
2027	2039	WHILE without WENDエラー
202A	203C	WEND without WHILEエラー
202D	203F	UNTIL without REPEATエラー
203D	2042	RETURN without GOSUBエラー
2033	2045	Can't continueエラー
2036	2048	String too longエラー
2039	204B	Unprintable error
203C	204E	Bad file modeエラー
203F	2051	No RESUMEエラー
2042	2054	Line buffer overflowエラー
2045	2057	Division by zeroエラー
2048	205A	Overflowエラー
204B	205D	Too complexエラー
204E	2060	Type mismatchエラー
2051	2063	FOR without NEXTエラー
2054	2066	Syntax error
2057	2069	Missing Operandエラー
205A	206C	Reserved featureエラー
205D	206F	Illegal function callエラー
2061	2073	ERRORエントリー
2064	2076	エラー共通処理
20ED	210E	エラー・トラッピング実行
2118	2139	16進→10進文字列変換
2128	2149	ENDエントリー
212F	2150	AUTOエントリー
217B	219F	AUTOフラグをチェックし、AUTO実行
2183	21A7	フリー・エリア先頭アドレス、BASIC用SP初期値をセット
218B	21AF	プログラムのNEW
21A0	21C4	NEWエントリー
21A7	21CB	NEW ON処理
21D2	21F6	NEW実行
21E3	2207	MAXFILESエントリー
21EE	2212	CLRエントリー
2254	2278	TRONエントリー
2255	2279	TROFFエントリー
225A	227E	TRON時、行番号表示
2272	2296	OPTIONエントリー
2276	229A	OPTION BASE処理
2297	22BB	OPTION SCREEN処理
22A6	22CA	LINPUT、LINE INPUTエントリー
2326	234A	各種INPUT文でのプロンプト表示
2335	2359	INPUTエントリー
23AB	23CF	INPUT等共通入力デバイス・セレクト
2496	24C1	カンマ、コロン、行エンドのチェック
249E	24C9	関数と同じ名前をもつステートメントの実行
24BE	24E9	CMT文（CMT＝）
24D1	24FC	イコール・チェック
24E9	2514	DATE$代入（DATE$＝）
2554	257F	，チェック（DE＝ポインタ）
2559	2584	：チェック（DE＝ポインタ）
255E	2589	／チェック（DE＝ポインタ）
2566	2591	DAY$代入（DAY$＝）
2595	25C0	TIME代入（TIME＝）

テープ BASIC	ディスク BASIC	機　　　　　能
25C6	25F1	TIMEとTIME$の場合分け（$チェック）
25CB	25F6	TIME$代入（TIME$＝）
2614	263F	10進キャラクタ・コード（Aレジ）→数値変換（Aレジ）
2683	26AE	MEM$代入（MEM$（，）＝…）
26AE	26D9	DEFエントリー
26CB	26F6	DEFUSR処理
26E4	270F	MID$代入
274C	2777	ただのRESTORE処理
2768	2793	RESTOREエントリー
276A	2795	RESTORE行番号（ラベル）処理
279F	27CA	READエントリー
28D1	28FC	“）”のチェック（HL＝ポインタ）
2DFD	2E2B	CALLエントリー
2E1F	2E4D	RANDOMIZEエントリー
2E39	2E67	DATA、LABELエントリー
2E45	2E73	“，”のチェック（HL＝ポインタ）
2E48	2E76	Aレジスタが“，”であることをチェック
2E4F	2E7D	SWAPエントリー
2EBC	2EEA	OUTエントリー
2ECA	2EF8	CLEARエントリー
2ECD	2EFB	LIMITエントリー
2EF7	2F25	DELETEエントリー
2F09	2F37	RENUMエントリー
2FB4	2FE2	行番号→アドレス変換
2FC1	2FEF	アドレス→行番号変換
302E	305C	行番号サーチ
305A	3088	開始行、終了行の行番号を得る（LIMIT、DELETEなど）
30A7	30D5	EDITエントリー
3101	3133	UNTILエントリー
3149	317B	RETURNエントリー
317D	31AF	WHILEエントリー
31B0	31E2	WENDエントリー
31EB	321D	REPEAT without UNTILエラー
3222	3254	REPEATエントリー
322A	325C	（UNTILと対の）REPEAT処理
3247	3279	REPEAT OFF処理
3248	327A	REPEAT ON処理
324D	327F	GOSUBエントリー
326E	32A0	ON KEY GOSUB実行
32A8	32DA	ラベルジャンプ用ラベルサーチ
32BB	32ED	ラベル・サーチ
3348	337A	RESUMEエントリー
3371	33A3	ON ERROR GOTO処理
33C4	33F6	ON ERROR GOTO実行
33DA	340C	ONエントリー
33E6	3418	ON～GOTOなど
3448	347A	GOエントリー
3456	3488	GOTOエントリー
3485	34B7	ON KEY GOSUB処理
34E0	3512	IFエントリー
34EC	351E	IFで真のときのTHEN処理
3500	3532	IFで偽のときのELSE処理
3520	3552	1ワード・サーチ（定数コードはスキップする）
3574	35A6	DEFINTエントリー
3577	35A9	DEFSTRエントリー
357A	35AC	DEFSNGエントリー
357D	35AF	DEFDBLエントリー
357F	35B1	DEF～共通処理

テープ BASIC	ディスク BASIC	機　　　能
35BD	35EF	WAITエントリー
3632	3664	POKEエントリー
3647	3679	POKEⓐエントリー
366B	369D	LOCATE、CURSORエントリー
3694	36C6	WIDTHエントリー
369C	36CE	WIDTH80処理
36A1	36D3	WIDTH40処理
36A6	36D8	WIDTH40用ワークエリア変更
36AC	36DE	WIDTH80用ワークエリア変更
3772	37A4	WINDOWエントリー
3840	3872	5バイト転送（単精度計算用）
3879	38AB	GETⓐ、PUTⓐ等用座標計算
3881	38B3	ビューポートを考えない座標計算
388A	38BC	PSETなど汎用の座標計算
38D9	390B	CONSOLEエントリー
3906	3938	ただのCONSOLE処理
390D	393F	CONSOLEの、xパラメータ初期化
393C	396E	KEY LLISTエントリー
393D	396F	KEY LIST、KLISTエントリー
393E	3970	KEY LIST関係共通処理
3A01	3A33	ON、OFF、STOPチェック
3A13	3A45	KEY ON、KEY OFF、KEY STOPエントリー
3A32	3A64	KEY、DEFKEYエントリー
3A45	3A77	KEY n エントリー（$1 \leqq n \leqq 10$）
3A80	3AB2	KEY 0 エントリー
3AA5	3AD7	COLORエントリー
3AC6	3AF8	ただのPALET処理（パレット初期化）
3AD4	3B06	パレット設定ルーチン
3AE6	3B18	PALETエントリー
3AEB	3B1D	PALET処理
3B23	3B55	PRWエントリー
3B2B	3B5D	プライオリティ設定ルーチン
3B38	3B6A	垂直同期（PV-Sync）待ち
3B42	3B74	PRESETエントリー
3B4D	3B7F	PSETエントリー
3B5C	3B8E	PSETモード・セット
3B62	3B94	PSET、SET、XOR等モード・セット
3B6B	3B9D	グラフィック座標→実アドレス変換
3BAF	3BE1	座標値、パレット・コードを得る（PSET等用）
3BD1	3C03	テキスト画面色コードをグラフィック用色コードとして設定
3BD8	3C0A	ただのSCREEN処理（グラフィック表示なし）
3BFD	3C2F	SCREEN、GRAPHエントリー
3C65	3C97	CLS n （$0 \leqq n \leqq 3$）処理
3C78	3CAA	ただのCLS処理
3C7C	3CAE	CLS 4 処理
3C82	3CB4	CLSエントリー
3D26	3D58	スペース読みとばし、コロン、エンドのチェック（HL＝ポインタ）
3D30	3D62	コロン、カンマ、エンドマーク（00H）のチェック（Aレジスタ対象）
3D3C	3D6E	LINEエントリー
3D47	3D79	（グラフィック画面での）LINEエントリー
3F80	3F83	LINEのB（BOX）用サブルーチン
3FD6	4009	LINEのBF（BOX FULL）用サブルーチン
40B6	40E9	PSETサブルーチン
4106	4139	テキスト画面へのPSET（テキスト画面のLINE用）
4137	416A	DEFCHR$エントリー
416A	419D	CREVエントリー
4178	41AB	アトリビュート用共通ルーチン
4181	41B4	CFLASHエントリー

テープ BASIC	ディスク BASIC	機　　　能
4191	41C4	CGENエントリー
41AF	41D4	CSIZEエントリー
41B2	41E5	CIRCLEエントリー
41B5	41E8	POLYエントリー
41B7	41EA	CIRCLE、POLY共通処理
44E7	451A	MUSIC、TEMPO、PLAYエントリー
4513	4546	TEMPO処理
453A	456D	音楽演奏
475D	4790	SOUNDエントリー
4771	47A4	PSGへのデータ出力（D=レジスタ番号、E=データ）
477A	47AD	BEEPエントリー
477C	47AF	ただのBEEP処理
478D	47C0	BEEP 1処理
47AC	47DF	BEEP 0処理
47C1	47F4	PAINTエントリー
498C	49BF	POINT値を求める。
4A7D	4AB0	カラーコード（0～7）を得る。
4A86	4AB9	EJECTエントリー
4A88	4ABB	CSTOPエントリー
4A8B	4ABE	FASTエントリー
4A8E	4AC1	REWエントリー
4A98	4ACB	APSSエントリー
4AA1	4AD4	APSS −n の時の処理
4AAA	4ADD	カセット制御命令共通処理
4AD8	4B0F	カセット走行チェックとサブCPU出力準備待ち
4AE5	4B1C	CANVASエントリー
4AF9	4B30	LAYERエントリー
4BEA	4C21	SCROLLエントリー
4C24	4C5B	SCROLL 0処理
4C2A	4C61	EFECTエントリー（Syntax errorとなる。）
4C2D	4C64	KBUFエントリー
4C3F	4C76	GETエントリー
4C44	4C7B	GET@エントリー
4E1A	4E57	PUTエントリー
4E1F	4E5C	PUT@エントリー
4F39	4F76	POSITIONエントリー
4F45	4F82	PATTERNエントリー
50AC	50E9	HCOPYエントリー
50B1	50EE	HCOPY 0 − 4 の判別
511D	515A	グラフィック印字終了（BREAK、終了時処理）
512B	5168	LPRINT CHR$（27、37）を実行。
513F	517C	HCOPY 0 処理
5145	5182	HCOPY 3 処理
514A	5187	HCOPY 2 処理
514F	518C	HCOPY 1 処理
5152	518F	HCOPY用共通ルーチン
5167	51A4	GRAM読み出し（WIDTH80用）
5174	51B1	GRAM読み出し（WIDTH40用）
51AF	51EC	ビットパターン転置処理
51CC	5209	GRAMパターン拡大処理
51F4	5231	HCOPY 4 処理
51F7	5234	ただのHCOPY処理（テキスト画面ハードコピー）
522A	5267	RAMCG読み出し } ハードコピー用
524F	528C	ROMCG読み出し } ハードコピー用
5272	52AF	1行の下半分出力パターン作成 } ハードコピー用
527C	52B9	1行の上半分出力パターン作成 } ハードコピー用
5293	52D0	BOOTエントリー
52A1	52DE	ASKエントリー

テープ BASIC	ディスク BASIC	機　　　能
52CB	530B	CLICKエントリー
52DA	531A	TVPWエントリー
52F3	5333	CHANNELエントリー
52FF	533F	VOLエントリー
5331	5371	CRTエントリー
536F	53AF	Format overエラー
5374	53B4	整数→実数変換
539F	53DF	DE＝DE＋8（倍精度ポインタ用）
53A7	53E7	倍精度の1を設定（DE＝ポインタ）
53AA	53EA	8バイト転送（DE＝転送先ポインタ、HL＝ソース・ポインタ）
53B0	53F0	数字キャラクタ（0～9）のチェック
53B7	53F7	英小文字→英大文字変換（対象は（DE））
53B8	53F8	英小文字→英大文字変換（Aレジスタ対象）
53C1	5401	変数名チェック（変数名として可能ならCy＝0）
53D9	5419	CDBLエントリー
53F6	5436	CSNGエントリー
540D	544D	倍精度→単精度変換
5440	5480	2進数文字列（&B）→2バイト整数変換
544C	548C	8進数文字列（&O）→2バイト整数変換
5458	5498	16進数文字列（&H）→2バイト整数変換
5462	54A2	整数変換ルーチン共通処理
5498	54D8	2進数キャラクタのチェック
549F	54DF	8進数キャラクタのチェック
54A6	54E6	16進数キャラクタのチェック
54B6	54F6	実数値の外部表現（10進文字列）→内部表現変換
5653	5693	実数値の内部表現→外部表現（10進文字列）※変換ワークに格納
56E3	5723	整数（HL＝ポインタ）をバッファに転送
56F8	5738	10進数文字列（0～65535。HL＝ポインタ）をバッファに転送。
5742	5782	浮動小数点→文字列変換（PRINT USING用）
591C	595C	浮動小数点→文字列変換（LIST時専用）
5A50	5A90	CINTエントリー
5A8E	5ACE	整数化（BASICのFIX相当）
5B20	5B60	小数部取り出し（BASICのFRAC相当）
5B58	5B98	内部表現での10倍（HL＝ポインタ）
5B65	5BA5	実数→2バイト整数変換（HLにセット。オーバーフローはCy＝1）
5B6F	5BAF	実数→2バイト整数変換（HLにセット。オーバーフローはエラー）
5C3A	5C7A	HLの内容を10進文字列変換（行番号など用）

インタプリタ・ブロック(2)〔ファイル処理中心〕

テープ BASIC	ディスク BASIC	機　　　能
6719	5CBB	DEVI$エントリー
676D	5D0F	DEVO$エントリー
67DF	5D83	LPOSエントリー
67E7	5D8B	POSエントリー
67FE	5DA2	POS（0）エントリー
680E	5DB2	DEVFエントリー（テープ版ではエラー）
680E	5DFE	FPOSエントリー（　〃　）
680E	5E18	LOCエントリー（　〃　）
680E	5E23	LOFエントリー（　〃　）
6811	5E3D	DEVICEエントリー
683D	5E69	FIELDエントリー（テープ版ではエラー）
――	5ED6	GET処理
――	5EED	PUT処理
――	602B	ATTR$エントリー
683D	6093	SETエントリー（テープ版ではエラー）

テープ BASIC	ディスク BASIC	機　　能
――	6097	SET〝ファイルディスクリプタ〟処理
――	60D9	SET ♯n（デバイス番号）処理
6844	6110	KILLエントリー
684C	6118	KILL ♯n（デバイス番号）処理
6862	612E	KILL〝ファイルディスクリプタ〟処理
6871	613D	SEARCHエントリー
68F2	61BE	LLISTエントリー
68F7	61C3	LISTエントリー
6922	61EE	アスキー・セーブ（SAVE、A）処理
69A9	6275	BREAKチェック
69E9	62B5	LOAD?　処理
69EA	62B6	VERIFY処理
6A3C	6308	LOADM処理
6AA3	636F	RUN〝ファイル名〟処理
6AAE	637A	LOADエントリー
6ADD	63A9	CHAINエントリー
6B55	6423	MERGEエントリー
6BC5	6495	SAVEM処理
6C04	64D4	SAVEエントリー
6C77	6547	CLOSEエントリー
6C96	6566	すべてのファイルのCLOSE処理
6CA3	6573	1つのファイルのCLOSE処理
6CBF	658F	EOFエントリー
6CE7	65BC	OPENエントリー
6D46	6627	ファイル・バッファのアドレスを得る。
6D78	6659	ファイル・ディスクリプタを得る。
6E07	66E7	INITエントリー
6E37	6718	LFILESエントリー
6E38	6719	FILESエントリー
――	672F	RSETエントリー
――	675C	LSETエントリー
――	679C	NAMEエントリー
6E4E	6822	デバイス処理ルーチンへのジャンプ
6E63	6837	〝Are you sure? (y or n)〟と聞く。

インタプリタ・ブロック(3)〔デバイス処理中心〕

テープ BASIC	ディスク BASIC	機　　能
5C7B	6876	Out of tapeエラー（カセットEJECTつき）
5C7F	687A	Out of tapeエラー（表示のみ）
5C82	687D	Device I/O error
5C85	6880	Device offlineエラー
5C88	6883	Input past endエラー
5C8B	6886	File not openエラー
5C8E	6889	File not foundエラー
5C91	688C	Already openエラー
5C94	688F	Bad file numberエラー
5C97	6892	Bad recordエラー
5C9A	6895	Bad screen modeエラー
5C9D	6898	Bad file descripterエラー
5CA0	689B	Bad file modeエラー
5CA3	689E	Tape read error
5CA6	68A1	Write protectedエラー
――	68A4	FIELD overflowエラー
5CA8	68A6	カセットを停止し、エラー表示
5CB2	68B3	マシン語ファイルのチェック

テープ BASIC	ディスク BASIC	機　　　能
5CB6	68B7	Bad allocation tableエラー
5CEA	68EB	ファイル属性 "I" のチェック
5CEE	68EF	ファイル属性 "O" のチェック
5CFB	68FC	デバイスSCRのPUT、SAVE
5D01	6902	デバイスCRTのPUT
5D07	6908	デバイスLPTのPUT
5D33	6934	デバイスCRTのPOS
5D67	6968	デバイスCRTへの改行出力
5D74	6975	デバイスCRTへの1文字出力
5DDE	69DF	デバイスKEYからの1文字入力
5E13	6A14	デバイスLPTのPOS
5E47	6A48	デバイスCASのVERIFY
5E4D	6A4E	デバイスCASのGET
5E60	6A64	デバイスCASのLOAD
5E66	6A6A	デバイスCASのPUT
5E6D	6A71	デバイスCASのSAVE
5E73	6A77	デバイスCASのCLOSE
5E93	6A9B	デバイスCASのKILL
5E9E	6ADD	デバイスCAS、MEM、EMM、ディスクからの1行入力※
5EB7	6AF6	デバイスCAS、MEM、EMM、ディスクからの1バイト入力※
5ECF	6B0E	デバイスCAS、MEM、EMM、ディスクのEOF※
602A	6C7D	デバイスCAS、MEM、EMM、ディスクのPOS※
602E	6C81	デバイスCAS、MEM、EMM、ディスクのTAB※
603B	6C8E	デバイスCAS、MEM、EMM、ディスクへの改行出力※
6042	6C95	デバイスCAS、MEM、EMM、ディスクへの1バイト出力※
		※以上で、テープ版ではディスクのサポートはなし。
6079	6CCC	デバイスCASのOPEN
6105	6D58	デバイスCASのINIT
6115	6D68	テープを巻き戻し、状態のチェック
――	6D7D	カセット停止後、BREAK表示
6125	6D85	時間待ち。
6132	6D92	デバイスCASのFILES
6146	6DA6	カセット関係エラー処理
6159	6DB9	カセット用FILES 1行表示
6163	6DC3	画面／プリンタへのFILES 1行表示
		（CAS、MEM、EMM、ディスク共用）※
6205	6E65	年月日データ・セット
6228	6E88	エンドマーク（00H）のセット
622D	6E8D	Aレジスタ→16進2文字（DE＝ポインタ）
623D	6E9D	月・曜日のデータ分離後、セット
6258	6EB8	時・分データ・セット
62C6	6F26	デバイスMEMのGET
62C8	6F28	デバイスMEMからのリード・データ
62D6	6F36	デバイスMEMのPUT
62D8	6F38	デバイスMEMへのライト・データ
62E6	6F49	デバイスMEMのVERIFY
62F9	6F5A	デバイスMEMの動作テストとアクセス・アドレス計算
631F	6F82	GRAM接続テスト
――	6F91	Bad recordエラー
632E	6F94	Device fullエラー
6363	6FCD	デバイスEMMの動作テストとアクセス・アドレス計算
63A5	700F	デバイスEMMへのアクセス・アドレス出力
63B0	701A	デバイスEMMのPUT
63B2	701C	デバイスEMMへのライト・データ
63BF	702C	デバイスEMMのGET
63C1	702E	デバイスEMMからのリード・データ
63CE	703B	デバイスEMMのVERIFY
63FC	705D	MEM、EMM、ディスクへのライト・データ※

テープ BASIC	ディスク BASIC	機　　　能
6402	7063	MEM、EMM、ディスクからのリード・データ※
6408	7069	MEM、EMM、ディスクのVERIFY※
640C	706D	上の共通処理（ジャンプ・テーブル・サーチ）
6425	7086	MEM、EMM、ディスクのFILES※
6459	70D0	MEM、EMM、ディスクのINIT※
6467	70DE	ディレクトリ初期化サブルーチン
6477	70EE	キー入力バッファ256 バイト・クリアルーチン
64AF	7126	FAT初期化サブルーチン
64BA	7131	MEM、EMM、ディスクのKILL
———	7196	Write protectのチェック
64DF	719D	ファイル属性のチェック
64E8	71A6	MEM、EMM、ディスクのOPEN※
64FF	71C2	"I" モードOPEN
———	7207	"A" モードOPEN
———	7291	"R" モードOPEN
653D	729F	"O" モードOPEN
65AB	7317	FAT内の00H→80H変換
6627	73A1	使用ディレクトリ終了時処理
662E	73A8	MEM、EMM、ディスクへのSAVE※
668D	7407	MEM、EMM、ディスクからのLOAD※
6702	747C	デバイス番号、ユニット番号バッファへのデータ・セット
		※以上でテープ版ではディスクのサポートはなし。
———	7500	ディスク・ドライブ電源ONチェック
———	751E	指定ドライブのモーターONとヘッド選択
———	7540	ディスク挿入のチェック
———	7551	リストア後、Dレジスタのトラックまでヘッドをシーク
———	756E	トラック番号、ポインタをHLにセット
———	757D	Dレジスタのトラックまでヘッドをシーク。
———	7592	FDCのBusyチェック
———	75A3	レコード番号→物理アドレス変換
———	75D1	ディスクからの1セクタ・リード
———	75D3	ディスクからのリード・データ
———	75FD	ディスクからの読みとり時、Not Ready処理
———	7607	ディスクからの読みとり時、エラー処理
———	7612	次のセクタを読むための処理
———	7620	次の物理アドレスを計算。
———	763C	FDCへのコマンド出力（E=セクタ番号、A=コマンド）
———	764C	指定ドライブのモーターOFF
———	7658	ディスクへの1セクタ・ライト
———	765A	ディスクへのライト・データ
———	768D	ディスクへの書き込み時、Not Busy処理
———	76A1	ドライブモーターOFF後、Device I/O error
———	76A7	ディスクへの書き込み時、エラー処理
———	76AC	次のセクタへ書き込むための処理
———	76BA	VERIFY準備処理
———	76CA	ディスクのVERIFY
———	76F6	VERIFY時、Not Ready処理
———	76FF	VERIFY時、エラー処理
———	770B	次のセクタのVERIFYのための処理
———	771B	Write protected fileでないことのチェック

インタプリタ・ブロック(4)

テープ BASIC	ディスク BASIC	機　　能
6ED6	7733	文字列→中間コード変換
6F45	77A2	予約語ワード・テーブル・サーチ
702F	788C	ワード・テーブルをサーチし、対応する中間コードを(HL)に格納。
7035	7892	中間コード・カウンタを＋1しながら、ワード・テーブルの文字列と比較。
7040	789D	中間コード・サーチ
7051	78AE	次の語まで読みとばし
7059	78B6	中間コード決定
705F	78BC	中間コードをバッファに格納
7076	78D3	スペース・コード転送（DE＝ソース、HL＝転送先）
70FF	795C	中間コード→文字列変換
72BE	7B1B	HLの内容を8進文字列に変換（DE＝ポインタ）
72F5	7B52	HLの内容を16進文字列に変換（DE＝ポインタ）
731A	7B77	HLの内容を2進文字列に変換（DE＝ポインタ）
7331	7B8E	10進文字列（DE＝ポインタ）→2バイト整数変換（BCにセット）
7344	7BA1	10進文字列（DE＝ポインタ）→2バイト整数変換（HLにセット）
7738	7F95	0～255のオペランドを得る。
773B	7F98	DEが0～255であることをチェックし、Aに値を移す。
7742	7F9F	DEに整数を得る（関数用）。
774D	7FAA	DEに整数を得る（汎用）。
775C	7FB9	DEに文字列格納アドレス、Aに文字数を得る。
7766	7FC3	ストリング・ディスクリプタ（3バイト）→文字列格納アドレス、文字数変換
7774	7FD1	汎用総合的演算ルーチン
7787	7FE4	EQV処理
77B7	8014	IMP処理
77D9	8036	XOR処理
77F9	8056	OR処理
7819	8076	AND処理
783A	8097	NOT処理
7875	80D2	＜処理
7890	80ED	＞処理
789B	80F8	＝処理
78A4	8101	＞＝、＝＞処理
78AD	810A	＝＜、＜＝処理
78B4	8111	＜＞、＞＜処理
78C6	8123	－処理（減算）
78D6	8133	＋処理（加算）
78EA	8147	MOD、￥処理（整数除算）
7916	8173	MODのみ実行
793B	8198	＊処理（乗算）
794B	81A8	/　処理（除算）
7965	81C2	符号の－処理
7992	81EF	^処理（ベキ乗）
79F4	8251	0～9の定数処理
7A04	8261	ポインタHLを＋1後、スペース・コード読みとばし
7A05	8262	スペース・コード読みとばし（HL＝ポインタ）
7A0F	826C	整数定数処理
7A3A	8297	実数定数処理
7A4F	82AC	ストリング定数処理
7A78	82D5	“（”のチェック
7A7C	82D9	“）”のチェック
7A88	82E5	定数π処理
7A9D	82FA	実変数、仮変数、関数名の場合わけ
7AB3	8310	(実)変数処理
7AC2	831F	FN内の仮変数処理
7B6A	83C7	関数中間コード処理
7B85	83E2	INT～SQR、ATN～STRIG、STR$～KANJI$中間コード処理

テープ BASIC	ディスク BASIC	機　　　　能
7BA0	83FD	PEEK ⓐエントリー
7BC5	8422	RNDエントリー
7BD1	842E	ただのRND処理
7BE2	843F	ASC～DEVF中間コード処理
7BFA	8457	ERR～DTL中間コード処理
7C03	8460	LEFT$～MIRROR$中間コード処理
7C15	8472	SIZEエントリー
7C24	8481	FREエントリー
7C3A	8497	CSRLINエントリー
7C42	849F	ERRエントリー
7C47	84A4	STRPTRエントリー
7C4D	84AA	DTLエントリー
7C53	84B0	STRIGエントリー
7C5A	84B7	STRIG(0)処理（スペースキー入力）
7C68	84C5	STRIG(1)、STRIG(2)処理（ジョイスティック入力）
7C6F	84CC	STICKエントリー
7C76	84D3	STICK(0)処理（テンキー入力）
7C83	84E0	STICK(1)、STICK(2)処理（ジョイスティック入力）
7C93	84F0	引数が0～2であることをチェック（STRIG、STICK用）
7C9D	84FA	ジョイスティック・ポートよりデータを得る。
7CA8	8505	リアルタイム・キースキャン（STRIG(0)、STICK(0)用）
7CB3	8510	ERLエントリー
7CD0	852D	CHR$エントリー
7D0F	856C	ストリング・ディスクリプタ（3バイト）をFACにセット（CHR$、STR$用）
7D34	8591	OCT$エントリー
7D3F	859C	BIN$エントリー
7D4A	85A7	HEX$エントリー
7D5D	85BA	ストリング・ディスクリプタ（3バイト）をFACにセット（汎用）
7D6D	85CA	FACの値を整数型にして、HLにセット。
7D78	85D5	HLの値を2進文字列16字に変換（DE＝ポインタ）
7D88	85E5	HLの値を8進文字列6字に変換（DE＝ポインタ）
7DA2	85FF	HLの値を16進文字列4字に変換（DE＝ポインタ）
7DC6	8623	MKI$エントリー
7DCB	8628	MKS$エントリー
7DD0	862D	MKD$エントリー
7DDC	8639	SPACE$エントリー
7E02	865F	CGPAT$エントリー
7E26	8683	KANJI$エントリー
7E9B	86F8	STR$エントリー
7EBE	871B	ASCエントリー
7ECA	8727	LENエントリー
7ED2	872F	VALエントリー
7EE8	8745	CVIエントリー
7EEC	8749	CVSエントリー
7EF0	874D	CVDエントリー
7F07	8764	VARPTRエントリー
――	877D	ファイル用VARPTR
7F19	8798	CMTエントリー
7F20	879F	ただのCMT処理
7F35	87B4	CMT（　）処理
7F61	87E0	LEFT$エントリー
7F79	87F8	RIGHT$エントリー
7F97	8816	MID$エントリー
7FCB	884A	MID$（，）のときの処理（文字数省略時）
7FEC	886B	DAY$エントリー
7FFD	887C	DATE$エントリー
800C	888B	TIME、TIME$エントリー
8012	8891	TIME$処理

テープ BASIC	ディスク BASIC	機　　　　能
8026	88A5	TIME処理
803D	88BC	INKEY$エントリー
8046	88C5	INKEY$（　）処理
8057	88D6	ただのINKEY$処理
807D	88FC	MEM$エントリー
80A9	8928	CHARACTER$エントリー
80B4	8933	SCRN$エントリー
810E	898D	POINTエントリー
8135	89B4	STRING$エントリー
818E	8A0D	INSTRエントリー
821B	8A9A	HEXCHR$エントリー
82A6	8B25	RIGHT$、LEFT$、MID$用サブルーチン
82BA	8B39	MIRROR$エントリー
82E1	8B60	USRエントリー
831B	8B9A	DEFUSRテーブルのアドレス計算
8328	8BA7	USR n の n をチェック
――	8BB5	CALCエントリー
8336	8BFD	ポインタHLを＋1後、スペース読みとばし、"（" チェック
8337	8BFE	スペース読みとばし、"（" チェック（HL＝ポインタ）
833A	8C01	Aレジスタが"（"であることをチェックし、HLを＋1する。
8341	8C08	Aレジスタが"）"であることをチェックし、HLを＋1する。
8367	8C2E	ワークエリア初期化後、変数アドレス計算。
836A	8C31	変数アドレス計算
8466	8D2D	（DE）→（HL）にBレジスタのバイト数転送後、メモリーチェック
8475	8D3C	メモリーチェック（SP－200H＜DE＋1ならエラー）
8483	8D4A	INPUT、INPUT$エントリー
848A	8D51	INPUT$処理
849E	8D65	ファイルからのINPUT$処理
869B	8F66	Reserved featureエラー
879F	906A	DIMエントリー
87B7	9082	DEF FNエントリー
8820	90EB	FNエントリー
88B4	917F	FACの型が文字列型であるかチェック
88BD	9188	2バイト整数加算　HL←HL＋DE
88C1	918C	2バイト整数乗算　HL←HL＊DE
88D1	919C	HL←HL＊256＋DE＊A
88E2	91AD	HL≧256ならエラー、HL≦255ならHL＝HL＊256
8905	91D2	符号ビットのチェック
890A	91D7	（HL）←（HL）＋1。浮動小数点表現での2倍。
890F	91DC	（HL）←（HL）－1。浮動小数点表現での½倍。
8917	91E4	FACのメモリー・オーバーのチェック
8928	91F5	FACの初期化
8943	9210	整数→単精度実数変換
8958	9225	BASICスタックの初期化
8967	9234	べキ乗（＾）計算
89FB	92DE	ABSエントリー
8A03	92E6	INTエントリー
8A21	9304	浮動小数点演算（1を引く）HL＝FACポインタ
8A27	930A	〃　　（1を足す）HL＝FACポインタ
8A2D	9310	〃　　（1と比較する）HL＝FACポインタ
8A33	9316	FIXエントリー
8A3C	931F	FRACエントリー
8A5C	933F	SQRエントリー
8A71	9354	SUMエントリー
8A96	9379	FACエントリー
8AEA	93CD	ATNエントリー
8BB2	9495	COSエントリー
8BCC	94AF	SINエントリー

テープ BASIC	ディスク BASIC	機　　　能
8CC6	95A9	TANエントリー
8DD7	96BA	SGNエントリー
8E00	96E3	RADエントリー
8E08	96EB	PAIエントリー
8E24	9707	INPエントリー
8E31	9714	PEEKエントリー
8E3A	971D	RNDエントリー
8E6C	974F	EXPエントリー
8F84	9867	LOGエントリー
905F	9942	実数剰余計算（SINなどで使用）
906A	994D	整数→単精度実数変換
9126	9A09	（HL）～（HL＋7）をクリア（FACクリア用）
9130	9A13	正負変換
914F	9A32	FACの型が文字列型でないことをチェック
9158	9A3B	浮動小数点減算〔(HL)←(HL)−(DE)〕
9161	9A44	浮動小数点加算〔(HL)←(HL)＋(DE)〕
9313	9BF6	文字列加算
9356	9C39	（HL）と（DE）の比較
9361	9C44	文字列比較
938E	9C71	整数比較
93A5	9C88	実数比較
956E	9E51	整数乗算
9619	9EFC	整数除算
963C	9F1F	HLDE（32ビット）÷BC→DE余りHL
9712	9FF5	浮動小数点乗算〔(HL)←(HL)＊(DE)〕
9807	A0EA	浮動小数点除算〔(HL)←(HL)／(DE)〕
9B28	A40B	プログラミング
9B9A	A47D	変数エリアの移動
9C0B	A4EE	DE行－HL行をDELETE
9C93	A576	IPLROMの（HL）をコール
9C9F	A582	Undefined functionエラー
9CA4	A587	JP（HL）

付録2　SHARP HuBASIC ワークエリア、データエリア一覧

- SHARP　HuBASICは
  テープBASIC　CZ－8CB01　Ver1.0
  ディスクBASIC　CZ－8FB01　Ver1.0
  の2つを対象としています。
- テープBASICとディスクBASICとで、アドレスの配列順が一部前後しますが、順序はディスクBASICに準拠しています。
- 本資料作成にあたり、雑誌「マイコン」1983年2月号および1984年3月号に掲載の高橋雄一氏の記事を参考にさせていただきました。

テープBASIC	ディスクBASIC	機　能
0006	0006	1行入力字数
0007	0007	WIDTH *n* の値（28Hか50H）
000E	000E	カーソル *x* 座標
000F	000F	カーソル *y* 座標
0016	0016	CONSOLE *y* 方向開始行
0017	0017	CONSOLE *y* 方向終了行
001E	001E	CONSOLE *x* 方向開始桁
001F	001F	CONSOLE *x* 方向終了桁
0026	0026	テキスト・アトリビュート・コード
0027	0027	テキスト画面クリア時に埋めるキャラクタ・コード
002E	002E	キー入力ワーク（ASCIIコード）
002F	002F	キー入力ワーク（状態コード）
0036	0036	BREAKキー・ワーク
0037	0037	キー入力フラグ（00H＝入力なし、FFH＝入力有）
0050	0050	未使用。
0051	0051	ON KEY GOSUBフラグ
0052	0052	モード2割り込みジャンプ・テーブル（2×10バイト）
0069	0069	コントロール・コード処理ジャンプ・テーブル（2×32バイト）
00A9	00A9	0ページ目用行連続フラグ・テーブル（26バイト）
00C3	00C3	1ページ目用行連続フラグ・テーブル（26バイト）
00DD	00DD	CRTC出力データ（20バイト）
		※内訳：00E9H　出力ページ・フラグ（2バイト）
		00EBH　入力ページ・フラグ（2バイト）
		00EDH　80桁用出力データ（4バイト）
00F1	00F1	アトリビュートVRAMエリア終了アドレス＋1（2バイト）
00F3	00F3	VRAMの余っているバイト数　CGアクセス用
00F4	00F4	テキストVRAMエリア終了アドレス＋1（2バイト）
00F6	00F6	パレット（Blue）設定データ
00F7	00F7	パレット（Red）設定データ
00F8	00F8	パレット（Green）設定データ
00F9	00F9	テキスト・プライオリティ設定データ
02DA	02DA	カーソル位置表示キャラクタ（CTRL＋GRAPH用）
0366	0366	キー入力リピート・フラグ

テープ BASIC	ディスク BASIC	機　　能
0507	0507	1行入力時、行挿入フラグ（11HかC3H）
0A8B	0A8B	OPTION SCREENパラメータ（GRAMクリア・フラグを兼ねる）
0E90	0E90	クリック音 ON/OFFフラグ
0E91	0E91	ベル出力時のPSGデータ（8バイト）
0E99	0E99	ベル出力時に今までのPSGデータを退避させるエリア（8バイト）
0EA1	0EA1	チェック・サム・バッファ（2バイト）
0EA3	0EA3	ESCフラグ
0EA4	0EA4	INSTモード・フラグ
0EA5	0EA5	KBUF ON/OFFフラグ
0EA6	0EA6	先行入力バッファ相対アドレス現在値（00H～3FH）
0EA7	0EA7	先行入力バッファ相対アドレス注目値（00H～3FH）
0EA8	0EA8	先行入力バッファ（64バイト）
0EE8	0EE8	割り込み発生フラグ（10バイト）
0EF2	0EF2	水平TAB設定フラグ・テーブル（80バイト）
0F42	0F42	ファンクション・キー・テーブル（16×10バイト）
1043	1043	モニター・コマンド処理ジャンプ・テーブル（3×10バイト）
119D	119D	モニター時、1回のダンプ・バイト数－1（2バイト）
11A2	11A2	モニター時、1行のダンプ・バイト数（初期値＝08H）
131B	131B	プリンタ・ヘッド位置ワーク（LPOS相当）
131D	131D	プリンタ水平TAB間隔ワーク（初期値＝07H）
145A	145A	メッセージ・データ〝Writing″, 0　※0はエンド・マーク
1462	1462	メッセージ・データ〝Found␣␣″, 0
146A	146A	メッセージ・データ〝Skip␣␣␣″, 0
1472	1472	モニター時（*Pによる）出力デバイス切り替えフラグ（0は画面、他はプリンタ）
1473	1473	メッセージ・データ〝ERR?″, 0
1480	1480	モニター用ファイル・バッファ（32バイト）

※ファイル・バッファ内訳

アドレス	バイト数	内容
1480H	1バイト	ファイル属性
1481H～148DH	13バイト	ファイル名
148EH～1490H	3バイト	ファイル名拡張子
1491H	1バイト	パスワード
1492H～1493H	2バイト	データ・ブロック長
1494H～1495H	2バイト	データ・ブロック開始アドレス
1496H～1497H	2バイト	データ・ブロック実行アドレス
1498H～149CH	5バイト	SAVE時タイマー・データ
149DH～149FH	3バイト	00H（未使用）

テープ BASIC	ディスク BASIC	機　　能
20F7	2118	メッセージ・データ〝Unprintable␣error″, 0
2109	212A	メッセージ・データ〝␣in␣″, 0
210E	212F	メッセージ・データ〝Ok″, 0
2111	2132	メッセージ・データ CHR$(7)+〝Break″, 0
28D5	2900	メッセージ・データ〝?␣″, 0
28D8	2903	定数：単精度60の内部表現（5バイト）。時刻計算用。
28DD	2908	定数：単精度86400の内部表現（5バイト）。1日の秒数。
28E2	290D	TIME文、TIME関数用ワーク・エリア（5バイト）
28E7	2912	FORの制御変数終了値バッファ（6バイト）
28ED	2918	INPUT文等用SPバッファ（2バイト）
28EF	291A	未使用（2バイト）
28F1	291C	DTL用データ・ポインタ（行番号）(2バイト）
28F3	291E	データ・ポインタ（アドレス）(2バイト）
28F5	2920	RESTOREフラグ
28F6	2921	通常予約語ワード・テーブル
2ACB	2AF6	拡張予約語ワード・テーブル
2BAC	2BD7	関数予約語ワード・テーブル
2CDF	2D0D	通常予約語エントリー・アドレス・テーブル
2D9F	2DCD	拡張予約語エントリー・アドレス・テーブル
3220	3252	未使用（2バイト）

テープ BASIC	ディスク BASIC	機　　　　能
35EF	3621	テンポラリー・ストリング・エリア終了アドレス＋1（2バイト）
35F1	3623	通常のSP退避エリア（2バイト）
35F3	3625	MAXFILESの値
35F4	3626	エラー・フラグ
35F5	3627	CONTフラグ
35F6	3628	ストリング・データ・バッファ先頭アドレス（2バイト）
35F8	362A	このフラグが00Hでないと、ダイレクト・ステートメント実行時にエラーとなる。
35F9	362B	トレース・フラグ
35FA	362C	オートスタート（LOAD,R）フラグ
35FB	362D	AUTOフラグ
35FC	362E	次行のアドレス（2バイト）
35FE	3630	エディタ注目行番号（2バイト）
3600	3632	AUTO時の増分（2バイト）
3602	3634	現在実行中の行番号（2バイト）
3604	3636	エラートラッピング・スタート・アドレス（2バイト）
3606	3638	エラー行の次行が始まるアドレス（2バイト）
3608	363A	エラー行番号（ERL相当）（2バイト）
360A	363C	エラー箇所アドレス（2バイト）
360C	363E	現在実行中のアドレス（2バイト）
360E	3640	エラー番号バッファ（ERR相当）
360F	3641	BREAK箇所アドレス（2バイト）
3611	3643	BREAK行番号（2バイト）
3613	3645	BREAK行の次行が始まるアドレス（2バイト）
3615	3647	未使用（1バイト）
3616	3648	FOR文での制御変数のストリング・ディスクリプタ（3バイト）
3619	364B	未使用（2バイト）
361B	364D	FOR文での制御変数の先頭1文字のASCIIコード
361C	364E	ディレイ・フラグ（PRINT文改行時などに入る）
361D	364F	OPTION BASE実行済フラグ
361E	3650	ON KEY GOSUB定義テーブル（2×10バイト）
3C42	3C74	SCREEN文処理ジャンプ・テーブル（2×4バイト）
40C6	40F9	グラフィック画面用カラー・コード
40DE		グラフィック処理用場合分け（2バイト）
412B	415E	グラフィック処理用 モード・テーブル（2×6バイト）
41B0	41E3	未使用（2バイト）
4431	4464	CIRCLE用SINテーブル（182バイト）
4719	474C	TEMPO換算値（2バイト）
471B	474E	音符長先頭バイト（3AH＝30H＋10）
471C	474F	音符長データ（10バイト）
4726	4759	PSG Aチャンネル・バッファ（9バイト）
472F	4762	PSG Bチャンネル・バッファ（9バイト）
4738	476B	PSG Cチャンネル・バッファ（9バイト）
4741	4774	（通常音用）分周比テーブル（2×7バイト）
474F	4782	（半音用）分周比テーブル（2×7バイト）
4EF6	4F33	PUTⓐモード処理テーブル
4F74	4FB1	未使用（4バイト）。
5135	5172	HCOPY 0～4処理サブルーチン・テーブル（2×5バイト）
533C	537C	PAINT用バックカラー（8バイト）
5344	5384	WINDOW使用中フラグ
5345	5385	ビューポート x 方向開始位置（X_s） 2バイト
5347	5387	ビューポート y 方向開始位置（Y_s） 2バイト
5349	5389	ビューポート x 方向の長さ（X_e-X_s） 2バイト
534B	538B	ビューポート y 方向の長さ（Y_e-Y_s） 2バイト
534D	538D	ウィンドウ x 方向開始位置（X_1） 5バイト
5352	5392	ウィンドウ y 方向開始位置（Y_1） 5バイト
5357	5397	ウィンドウ x 方向表示比（$\frac{X_e-X_s}{X_2-X_1}$） 5バイト
535C	539C	ウィンドウ y 方向表示比（$\frac{Y_e-Y_s}{Y_2-Y_1}$） 5バイト
5361	53A1	SCREEN用ワーク（0ページ用）

テープ BASIC	ディスク BASIC	機　　　能
5362	53A2	SCREEN用ワーク（1ページ用）
5363	53A3	CANVASフラグ（GRAM 1用）
5364	53A4	〃　　（GRAM 2用）
5365	53A5	〃　　（GRAM 3用）
5366	53A6	LAYERフラグ（テキスト用）
5367	53A7	〃　　（GRAM 1用）
5368	53A8	〃　　（GRAM 2用）
5369	53A9	〃　　（GRAM 3用）
536A	53AA	CSIZEフラグ
536B	53AB	メッセージ・データ "KEY", 0
5B9D	5BDD	定数：倍精度で指数表現にならない最大数＋1（＝1D+16)。8バイト
5BA5	5BE5	定数：倍精度の1000000000000000。8バイト。
5BAD	5BED	定数：〃　100000000000000　。8バイト。
5BB5	5BF5	定数：〃　10000000000000　。8バイト。
5BBD	5BFD	定数：〃　1000000000000　。8バイト。
5BC5	5C05	定数：〃　100000000000　。8バイト。
5BCD	5C0D	定数：〃　10000000000　。8バイト。
5BD5	5C15	定数：〃　1000000000　。8バイト。
5BDD	5C1D	定数：〃　100000000　。8バイト。
5BE5	5C25	定数：〃　10000000　。8バイト。
5BED	5C2D	定数：〃　1000000　。8バイト。
5BF5	5C35	定数：〃　100000　。8バイト。
5BFD	5C3D	定数：〃　10000　。8バイト。
5C05	5C45	定数：〃　1000　。8バイト。
5C0D	5C4D	定数：〃　100　。8バイト。
5C15	5C55	定数：〃　10　。8バイト。
5C1D	5C5D	定数：〃　1　。8バイト。
5C25	5C65	定数：〃　0.1　。8バイト。
5C2D	5C6D	定数：倍精度で指数表現にならない最小数 （＝0.0000000000000001）。8バイト。
5C35	5C75	定数：単精度で指数表現にならない最小数 （＝0.00000001）　。5バイト。
6E8A	685E	メッセージ・データ "Are␣you␣sure␣?␣（y␣or␣n)", 0
5CBA	68BB	デバイスSCRのテーブル（48バイト）
5D37	6938	デバイスCRTのテーブル（48バイト）
5DAE	69AF	デバイスKEYのテーブル（48バイト）
5DE3	69E4	デバイスLPTのテーブル（48バイト）
5E17	6A18	デバイスCASのテーブル（48バイト）
626E	6ECE	ファイル属性用メッセージ "Asc", 0
6272	6ED2	〃　"Bin", 0
6276	6ED6	〃　"Dir", 0
627A	6EDA	〃　"Bas", 0
627E	6EDE	曜日用メッセージ　"SUN"
6281	6EE1	〃　"MON"
6284	6EE4	〃　"TUE"
6287	6EE7	〃　"WED"
628A	6EEA	〃　"THU"
628D	6EED	〃　"FRI"
6290	6EF0	〃　"SAT"
6293	6EF3	〃　"???"
6296	6EF6	デバイスMEMのテーブル（48バイト）
6333	6F9D	デバイスEMMのテーブル（48バイト）
63EA	704B	MEM、EMM、ディスクの「ライト・データ」用ジャンプ・テーブル（2×3バイト）
63F0	7051	〃　「リード・データ」用ジャンプ・テーブル（2×3バイト）
63F6	7057	〃　「ベリファイ」用ジャンプ・テーブル（2×3バイト）
6717	7491	デバイス・テーブル・エンド・マーク0000H（2バイト）
6EA2	7493	DEVICE文で指定したデフォルト・デバイス名（4文字＋":"の5バイト）
6EA7	7498	ファイル・ディスクリプタ用バッファ（32バイト）

テープ BASIC	ディスク BASIC	機　　　　能
6EC7	74B8	未使用（1バイト)。
6EC8	74B9	デバイス名バッファ（3バイト）
6ECB	74BC	未使用（2バイト)。
6ECD	74BE	デバイス・コントロール用バッファ（FDD制御など）
6ECE	74BF	デバイス番号バッファ (00H=SCR、01H=CRT、02H=KEY、 03H=LPT、04H=CAS、05H=MEM、 06H=EMM、07H=ディスク)
6ECF	74C0	LOADMで指定したロード先頭アドレスバッファ（2バイト）
6ED1	74C2	未使用（1バイト）
6ED2	74C3	ディスク・ドライブ0のトラック番号ワーク（テープBASICでは未使用）
6ED3	74C4	〃　1のトラック番号ワーク（テープBASICでは未使用）
6ED4	74C5	〃　2のトラック番号ワーク（テープBASICでは未使用）
6ED5	74C6	〃　3のトラック番号ワーク（テープBASICでは未使用）
――	74C7	デバイス「ディスク」(0:～3:）のテーブル（48バイト）
――	7724	メッセージ・データ "␣Clusters␣free", 0
736B	7BC8	通常エラー・メッセージ・テーブル
759A	7DF7	ファイル関係エラー・メッセージ・テーブル
7698	7EF5	関数予約語ジャンプ・テーブル
869E	8F69	未使用（5バイト)。
88EB	91B6	配列変数に格納されるデータのテキスト・ポインタ（2バイト）
88ED	91B8	未使用（2バイト）
88EF	91BA	配列定義済フラグ（00H=未定義、FFH=定義済）
88F0	91BB	OPTION BASEの値（0か1）
88F1	91BC	ジョイスティック変換用テーブル（2×8バイト）
――	91CC	CALC用ワークエリア（2バイト）
8901	91CE	仮変数の変数エリア・ポインタ（2バイト）
8903	91D0	仮変数への代入箇所のテキスト・ポインタ（2バイト）
8956	9223	未使用（2バイト)。
8AE6	93C9	ATN計算用浮動小数点データ（2バイト=7EH、4CH）
8AE8	93CB	ATN計算用浮動小数点データ（2バイト=80H、2BH）
8B7E	9461	定数：倍精度の$\sqrt{2}-1$（8バイト）
8CFF	95E2	SIN用定数データ（8×8バイト）
8D3F	9622	COS用定数データ（8×8バイト）
8D7F	9662	ATN用定数データ（8×10バイト）
8DCF	96B2	定数：倍精度の$\pi/4$（8バイト）
8E14	96F7	定数：倍精度のπ（8バイト）
8E1C	96FF	RAD用定数
9073	9956	定数：倍精度の$2/5$（8バイト）
907B	995E	定数：倍精度の$5/3$（8バイト）
9083	9966	LOG用定数（8×16バイト）
9103	99E6	定数：倍精度の$1/120$（8バイト）
910B	99EE	定数：倍精度のLOG(2)（8バイト）
9113	99F6	定数：単精度のLOG(2)（5バイト）
9118	99FB	定数：倍精度の1/LOG(2)（8バイト）
9120	9A03	定数：単精度のLOG(2)（5バイト）
9125	9A08	^（ベキ乗計算）用ワークエリア（変数型)(1バイト)
9CA5	A588	SIN等の計算用FACアドレス・テーブル（2×6バイト）
9CB1	A594	SIN等の計算用符号ワーク・エリア（1バイト）
9CB2	A595	RANDOMIZEワークエリア（2バイト；初期値=4193H）
9CB4	A597	EXP計算用ワークエリア（3バイト）
9CB7	A59A	LOG計算用ワークエリア（1バイト）
9CB8	A59B	各種変換用ワークエリア（51バイト） ※うち9CC9H～（ディスクA5ACH～）は 内部表現→外部表現変換用のワークエリア
9CEB	A5CE	LIST時の文字数カウンタ
9CEC	A5CF	四則演算用ワークエリア（12バイト）
9CF8	A5DB	FACの型（02H=整数、03H=文字列、05H=単精度、08H=倍精度）

テープ BASIC	ディスク BASIC	機　　　能
9CF9	A5DC	SWAP用などの汎用ワークエリア（8×3バイト）
9D11	A5F4	変換用ワークエリア（7バイト）
9D18	A5FB	DEFINT、SNG、DBL、STR用フラグ・テーブル （A～Zの26バイト分）
9D32	A615	DEFUSRテーブル（2×10バイト）
9D46	A629	変数エリア先頭アドレス（2バイト）
9D48	A62B	ファイル用ストリングバッファ先頭アドレス（2バイト） ※STRPTR相当
9D4A	A62D	テンポラリー・ストリング・エリア先頭アドレス（2バイト）
9D4C	A62F	BASIC用SP初期値（2バイト；初期設定＝FE00H） ※CLEARアドレスで設定される値－100H
9D4E	A631	ユーザー用マシン語フリーエリア先頭アドレス（2バイト） ※CLEARアドレス } LIMITアドレス } で設定される値
9D50	A633	IPL、モニター用ワークエリア先頭アドレス（2バイト） ※初期設定＝FF00H
9D52	A635	テキスト・エリア先頭アドレス（2バイト） ※NEW ONアドレスで設定される値 ※初期設定＝9FC5H（テープ版） ＝A8A8H（ディスク版）
9D54	A637	中間コードバッファ（258バイト） ※起動時には、起動メッセージが 格納されている。
9E56	A739	DATE用バッファ（1バイト）
9E57	A73A	DAY用バッファ（2バイト）
9E59	A73C	TIME用バッファ（3バイト）
9E5C ～9FC4	A73F ～A8A7	キー入力バッファ（361バイト）

付録3 I/Oマップ

※個別アクセスモードにおけるI/Oマップです。
※システムI/Oポート部分を中心にしてあります。
　VRAMに関しては第1章、GRAMに関しては第2章で詳述しました。

I/O アドレス（16進）

アドレス	ポート	備考
0000～0CFF	ユーザー用 I/O ポート	
0D00	外部メモリー（EMMボード）用 I/O ポート	
0E00	ROM BASIC カード (CZ-8RB) 用 I/O ポート	
0E80～0E82	漢字 ROM ボード (CZ-8KR) 用 I/O ポート	※0E80H、0E81H：データ入出力 ※0E82H：データ読み出しの開始・終了指示
0FF8～0FFC	ディスク・インターフェース用 I/O ポート (FDC)	※0FF8H：FDCのCR、STR 0FF9H：FDCのTR 0FFAH：FDCのSCR 0FFBH：FDCのDR 0FFCH：FDDのモーターON/OFF ヘッド、ドライブ指定
1000	パレット（B）設定ポート	※アドレスバスの上位8ビット A_{15}～A_8のみ有効。（1000～1300）
1100	パレット（R）設定ポート	
1200	パレット（G）設定ポート	
1300	テキスト・プライオリティ設定ポート	
1400	ROMCG アクセス・ポート	※アドレスバスの上位8ビット A_{15}～A_8、下位3ビット A_2～A_0が有効。 ※A_2～A_0によりフォント・パターンの横一列がアクセスされる。（1400～1700）
1500	PCG（B）アクセス・ポート	
1600	PCG（R）アクセス・ポート	
1700	PCG（G）アクセス・ポート	
1800	CRTC アクセス・ポート	※アドレスバスの上位8ビット A_{15}～A_8、最下位ビット A_0が有効。 ※1800H：CRTCのレジスタ選択 1801H：CRTCへのデータ出力
1900	サブ側8255 ポートA	※サブCPU 80C49とのデータ入出力ポート（キースキャン・ポートを兼ねる）
1A00	メイン側8255	※アドレスバスの上位8ビット A_{15}～A_8、下位2ビット A_1、A_0が有効。 ※1A00H：メイン側8255 ポートA 1A01H：〃 ポートB 1A02H：〃 ポートC 1A03H：〃 コントロール・レジスタ
1B00	PSGのアクセス・ポート（データ入出力）	※アドレスバスの上位8ビット A_{15}～A_8のみ有効。 ※ジョイスティック入力ポートを兼ねる。

アドレス	内容	備考
1C00	PSGのアクセス・ポート（レジスタ選択）	※アドレスバスの上位8ビット A_{15}～A_8 のみ有効。 ※PSGのレジスタ番号を出力。
1D00	IPLROMへのバンク切り替え用出力ポート	※アドレスバスの上位8ビット A_{15}～A_8 のみ有効。 ※データを出力すると、IPLROMのバンクに切り替わる。
1E00	メインRAMへのバンク切り替え用出力ポート	※アドレスバスの上位8ビット A_{15}～A_8 のみ有効。 ※データを出力すると、メインメモリーがすべてRAMに切り替わる。
1F00		
2000～27FF	アトリビュート VRAM	
3000～37FF	テキスト VRAM	
4000～7FFF	GRAM（B）	
8000～BFFF	GRAM（R）	
C000～FFFF	GRAM（G）	

参考文献

マイクロコンピュータ全般に関する文献

本書を執筆するに際して参照した文献、および、本書執筆後に発刊された文献のうち本書のテーマに添うものを選んで以下に掲げます。本文中で〔　〕つきで引用した番号は、以下の文献番号に対応しています。

Z80CPUのハードウェアおよび命令体系に関しては、次の本を参考にしました。

〔1〕 **図解マイクロコンピュータ　Z80の使い方**
(横田英一著。オーム社刊。)

〔2〕 **Z80アセンブラ言語入門 『マイコンピュータNo.7』**
(白鳥正美、額田忠之著。CQ出版社刊。)

〔3〕 **工作からプログラミングまで　Z80マイコン実習入門**
(横山直隆著。技術評論社刊。)

〔4〕 **Z－80マイコンの作り方《電子科学ブルーブックス》No.11**
(大川善邦著。廣済堂産報出版刊。)

次の2冊の本は、いずれも本書執筆後に発行されましたが、Z80CPUのプログラム技法の解説に富んだ良書なので掲げておきます。

〔5〕 **MSX FAN SERIES 3 マシン語入門（実践編）**
(渡辺卓也、樋口賢治著。エム・アイ・エー刊。)

〔6〕 **インターフェース別冊　Z80上級プログラミング**
――制御用を中心にした実践アセンブラ技法――
(西沢　昭著。CQ出版社刊。)

次の3冊は、CRT画面の表示原理を勉強する上で参考になりました。

〔7〕 **つくるシリーズ6　つくるCRTディスプレイ**
(トランジスタ技術編集部編。CQ出版社刊。)

〔8〕 **SCIENCE AND TECHNOLOGY 電子画像のはなし**
(木内雄二著。日刊工業新聞社刊。)

〔9〕 **マイクロコンピュータCRTディスプレイ技法《電子科学シリーズ》No.79**
(鈴木八十二著。廣済堂産報出版刊。)

周辺LSI（ペリフェラル）については次の文献を参照しました。

〔10〕 **インターフェースLSIの研究『マイコンピュータ No.2』**
(松本吉彦、額田忠之著。CQ出版社刊。)

〔11〕 **続インターフェースLSIの研究『マイコンピュータ No.8』**
(額田忠之、加藤大典、千葉憲昭著。CQ出版社刊。)

〔12〕 **ホビーエレクトロニクス 作れるマイコンインタフェース**
(矢野越夫著。日本放送出版協会刊。)

〔13〕 **作りながら学ぶ マイコン設計トレーニング**
(神崎康宏著。CQ出版社刊。)

〔14〕 '82 三菱半導体データブック
マイクロコンピュータ関連LSI編
(三菱電機刊。)

カセットテープのフォーマットについては次の記事が面白いです。

〔15〕 テープの物理フォーマット『The BASIC 1984年5月号』
(大場不労著。)

フロッピーディスクおよびFDCについては、次の文献を参考にしました。

〔16〕 最新フロッピ・ディスク装置とその応用ノウハウ
標準/ミニ/マイクロFDDシステムの基礎・設計・応用
(高橋昇司著。CQ出版社刊。)

〔17〕 ディスク百科　DOS & FLOPPY DISK SYSTEM
(柿園昭俊、橋口技研著。技術評論社刊。)

〔18〕 FMのディスク百科『The BASIC 1984年8月号』
(MAC．K　著)

〔19〕 MB8876A　MB8877A　ユーザーズマニュアル
(富士通株式会社刊。)

次の本は、マイクロコンピュータ全般について大変勉強になりました。とくにブートストラップの項は感激的です。

〔20〕 図解マイコンのプログラム　考え方からOSまで
(平松啓二、守屋慎次、斉藤　剛著。オーム社刊。)

マイクロコンピュータ関係の専門用語については、次の辞典を参照しました。

〔21〕 新マイコン用語辞典《電子科学ブルーブックス》No.4
(日本電気㈱電子デバイスグループ編著。廣済堂産報出版刊。)

〔22〕 マイコンOA辞典
(井上壽雄著。電波新聞社刊。)

あとがき

最初のX1（CZ－800C）の発表以来2年近くが経過しました。この間、X1C、X1D、X1C_s、X1C_kというX1の後継機種が続々と発表され、X1シリーズは8ビットパーソナルコンピュータとして確固たる位置を占めるに至りました。X1シリーズを活用するためのソフトウェアも、ゲームソフトを中心に充実してきています。

このようなX1シリーズをとりまく状況の中で、ユーザーとして敢えて1つの不満な点を指摘するとすれば、解説書の少なさがあげられます。X1シリーズの機種はいずれも、PCシリーズ・FMシリーズに優るとも劣らない性能を持っていますが、これら両シリーズの圧倒的な解説書の多さに比べると、X1シリーズの解説書はまだまだ不足しているといえるでしょう。とくに、マシン語に関する解説書はまだ数えるほどしかありません。

私は、X1により初めてパソコンの素晴らしさに触れたユーザーで、パソコン経験も2年に足りません。こんな私が本書のような本を書こうと思い起ったのも、上述の状況と無関係ではありません。もし私が、PCシリーズやFMシリーズのユーザーであったなら、豊富な良書がありますから、自ら筆をとってそれらに一冊を加えようとは思わなかったでしょう。

こうして、本書は「人気機種のわりに本が少ない」というX1シリーズ独特の状況により成立したのですが、反面、これこそが本書の特色ともいえると思います。従来、本書のような本は「解析マニュアル」として、専門家の方々により執筆されるのが通例でありました。これらの本はいずれも各機種の深いところまで解析してあり、熟読すると、その機種のユーザーならずとも学ぶ点がたくさんあります。私も、他機種用に書かれたこれらの「名著」を読んで、X1のマシン語をマスターすることができました。しかし、「解析マニュアル」は初心者向けではないために、勉強の過程で何回も壁にぶつかりました。とくに、専用LSIの機能や、コンピュータ独特の専門用語・概念の知識を前提としての説明には、理解までに多くの時間を要しました。

本書は、こうした私の「マシン語学習体験」を生かして執筆されています。ページ数の都合で、「Z80マシン語基礎編」を省かせていただきましたが、マシン語に興味を持ち、X1を実際にマシン語で動かしてみようとする時、ユーザーがぶつかるであろう壁には多くのページをあてて解説しました。とくに、X1のHu BASICの解析にあたり、初心者の私にとって最大の難関であった専用LSI（8255やFDC）の解説には力点を置いています。本書が読者の皆様に「X1マシン語活用コース」の近道を与えるとすれば、私にとって最高の喜びです。

私自身が詳しくないために、本書に盛り込むことのできなかった重要なテーマには、

①X1によるビデオ活用法
（デジタルテロッパーの活用や、コマ撮り録画など）
②X1によるシンセサイザー活用法
（MIDI規格の活用など）

などがあります。X1シリーズには、「VI」（ビジュアル・インテグレーション）というその設計思想からして、ビデオやシンセサイザーとのドッキングは最初から想定されているはずです。これらに堪能なユーザーの皆様、X1シリーズをさらに活用するためのノウハウを公開してください。X1シリーズに秘められた大きな可能性の扉を開くためには、幅広いユーザーの方々の奮起なくしてはあり得ません。

最後になりましたが、本書の出版を実現してくださった㈱産業報知センター（旧社名廣済堂産報出版）の皆様ならびに㈱モダンの皆様に感謝を捧げ筆をおきます。

著者記す。

索引

せ

そ

た

ち

つ

て

と

な

に

の

は

ひ

著者紹介

清水　保弘（しみず　やすひろ）

1954年，横浜市生まれ。東京大学理学部数学科卒業。現在，東京都立大学大学院に在学中。専攻は数学（幾何学，群論）。数学研究におけるパソコン利用の可能性について追究している。

編集/制作　株式会社 モダン

X1マシン語活用百科

定価2,500円

1984年12月15日　初版発行
1985年４月15日　再版発行
1986年１月10日　新装版発行

著　者　清　水　保　弘
発行者　三　好　哲　雄
発行所　秋葉出版株式会社
東京都千代田区神田和泉町１－１（郵便番号101）
電話 03（866）5491 代表・振替 東京 7-161626

〈検印廃止〉

印刷＝壮光舎印刷株式会社
製本＝秋元製本株式会社

万一，乱丁，落丁がございましたら書店または発行所でお取換えいたします。

ISBN4-87184-034-4

X1マシン語活用百科

- X1シリーズをマシン語で活用しようとする際にぶつかる難点を丁寧に解説。
- 画面表示、サウンド機能をはじめ、カセット、プリンタ、漢字ROM、フロッピーディスクなどの豊富な機能に対応。
- 周辺LSI、I/Oポートマップなど、ハードウェアにまでつっこんで詳しく説明。
- 巻末に、SHARP HuBASICエントリー・アドレス、ワークエリア、データエリアの一覧表とI/Oマップを掲載。

定価2,500円　ISBN4-87184-034-4 C3055 ¥2500E

X1マシン語活用百科

定価2,575円（本体2,500円）

■X１シリーズをマシン語で活用しようとする際にぶつかる難点をていねいに解説。

■画面表示，サウンド機能をはじめ，カセット，プリンタ，漢字ROM，フロッピーディスクなど豊富な機能に対応。

■周辺LSI，I／Oポートマックなど，ハードウェアにまでつっこんで詳しく説明。

■巻末に，SNARP HuBASICエントリー・アドレス，ワークエリア，データエリアの一覧表I／Oマップを掲載。

ISBN4-87184-034-4 C3055 P2575E

X1マシン語活用百科

シリーズ全機種対応!!

清水保弘著

秋葉出版